FutureTPM

D4.1

Threat Modelling & Risk Assessment Methodology

Project number:	779391
Project acronym:	FutureTPM
Project title:	Future Proofing the Connected World: A Quantum-Resistant Trusted Platform Module
Start date of the project:	1st January, 2018
Duration:	36 months
Programme:	H2020-DS-LEIT-2017

Deliverable type:	Report
Deliverable reference number:	DS-06-779391 / D4.1 / 1.0
Work package contributing to the deliverable:	WP 4
Due date:	Dec 2018 – M12
Actual submission date:	6th February, 2019

Responsible organisation:	UBITECH
Editor:	Sofianna Menesidou (UBITECH)
Dissemination level:	PU
Revision:	1.0

Abstract:	Deliverable D4.1 provides the details of the Risk Assessment (RA) methodology that will be followed in FutureTPM towards the design and implementation of a holistic RA framework capable of providing *vulnerability analysis* and *policy enforcement* during both **design-** and **run-time**. It also provides the analysis of the TPM commands that will be used as the baseline for our investigation (per reference scenario). Each reference scenario will focus on one main TPM functionality including *Sealing*, *Direct Anonymous Attestation* (DAA) and *Key Creation and Storage*.
Keywords:	Risk Assessment, Threat modelling, Vulnerability Analysis, Mitigation Enforcement, Control Flow Integrity, extended Berkeley Filters, Policy Enforcement

The project FutureTPM has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779391.

Editor

Sofianna Menesidou (UBITECH)

Thanassis Giannetsos (DTU)

Contributors (ordered according to beneficiary numbers)

Liqun Chen (SURREY)

Petros Mantos, Panagiotis Gouvas (UBITECH)

Roberto Jordaney, Daniele Sgandurra (RHUL)

Dimitris Panopoulos, Ioanna Michael, Alexandros Tsitsanis (SUITE5)

Christos Xenakis, Christoforos Ntantogian, Eleni Veroni, Nikolaos Koutroumpouchos (UPRC)

Roberto Sassu, Silviu Vlasceanu (HWDU)

Fanis Sklinos, Stratos Moros (INDEV)

Executive Summary

With the design and documentation of the FutureTPM Reference Architecture presented in D1.2 [1], the consortium identified the core services and components that need to be implemented towards the development of QR-based TPM environments for enhancing the security and privacy posture of cyber-physical systems. These fall into three main categories, namely the **Design and Risk Management** phases and the **Implementation** phase comprising the implementation and integration of the individual FutureTPM components. These services were defined in complete alignment with the derived functional and non-functional requirements, as documented in D1.1 [2], and constitute the main investigation points of the core technical work packages: WPs 3 and 4, and WPs 2 and 5, respectively.

Based on this comprehensive analysis, in this deliverable, we delve into the details of the Risk Management phase (WP4) where the focus is to identify and alleviate the complex threat landscape that the devices (e.g., processors, mobile devices, ASICS, etc.) hosting the TPM pose to the cryptographic applications. In many cases, the operation of such devices may leak sensitive information, which can be used to mount successful attacks to recover secret information. To mitigate such behavior, FutureTPM will start with the analysis and development of **run-time risk assessment and vulnerability analysis methodologies** of the existing TPM-based commodity systems and then enhance this analysis to the quantum resistance (QR) realm. It will consider a wide palette of threats and an identified segmentation of vulnerabilities for the TPM and the host device starting with the analysis of the core functionalities as have been identified in the context of the envisioned use cases (i.e., *Sealing*, *DAA* and *Key Creation and Storage*). This process is tailored to address the specific challenges of performing risk assessment and providing implementation guidance.

Risk Management refers to the process that allows security teams to assess, evaluate, measure and address potential security threats, to minimize the effect that they might pose to corporate and personal assets. Risk assessment (RA) is a process within risk management that deals with the identification of threats, and determines their probability of occurrence, and their resulting impact. A holistic risk assessment will focus on the TPM device itself, but also to the host device and all the remaining assets of the envisioned three reference scenarios. However, we will especially focus on the risk assessment of the TPM Software Stack (TSS) and TPM. In parallel, the consortium has worked towards defining the mapping between the project's risk assessment methodology and the reference scenarios, identifying how the risk assessment methodology will be applied based on the exact needs of each use case and the type of TPM environment that will be tested in each scenario.

The output of this investigation is the definition of the conceptual architecture of the FutureTPM Risk Assessment framework for supporting the *design*, *test* and *validation* of, not only, a QR TPM, but also a set of mutually interconnected components that will ensure security properties throughout the whole life cycle of cyber-physical systems. Details are presented on all aspects of the core services to be offered from **risk identification and quantification to run-time risk assessment and security policy enforcement**, including the interfaces exposed by the different components of the framework. This will steer the subsequent implementation of the specific integral components, namely the **Design-time Risk Assessment**, the **Policy Modelling and Specification**, the **Run-time Risk Assessment** and the **Dynamic Update of the Policy Models**.

In a nutshell, by leveraging the Risk Assessment component, a set of security policies will be defined in order to satisfy the security requirements of a given scenario, e.g., under what conditions can a certain private key sign some information. Ensuring and imposing the correct usage of this set of rules is the task of the Security Policy Enforcement module. The **Risk Assessment** phase will be executed during both the **design-** and **run-time**, to address initially

unknown threats. During the design-time, the risk assessment will be applied by security experts in order to provide a cartography of their TPM supported/enabled applications and services ecosystem. During the run-time, it will be applied towards the serialization of all information that is required to perform the re-calculation of relevant risks that may lead to a (possible) dynamic update of predefined security policies. The **Policy Enforcement** component will block unsafe usage of API depending on the provided policies, which will be elaborated to tailor the requirements of the three envisioned Reference Scenarios.

Overall, the purpose of deliverable D4.1 is to provide a reference document for the FutureTPM threat modelling and risk assessment methodology and to be used as the guide for the further development of the FutureTPM RA framework to be presented in the subsequent deliverables of WP4.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

The main goal of the FutureTPM project is to design a **Quantum-Resistant (QR) Trusted Platform Module (TPM)** by selecting, designing and developing QR algorithms suitable for inclusion in a TPM. In order to achieve this, apart from the appropriate QR algorithm selection, a systematically risk assessment is necessary to confirm the security and privacy of the QR TPM-based environment. In D1.2 [1] we have already identified the overall architecture of the FutureTPM as well as the components that comprise it, in complete alignment with the derived functional and non-functional requirements. More specifically, the FutureTPM architecture consists of three main components: a) the ***QR TPM***, b) the ***Risk Assessment*** and c) the **Security *Policy Enforcement*** components.

As a technical solution, it will provide *security*, *privacy* and *assurance* services to the deployed platforms, with a special focus on the potential threats that quantum computers pose when they become reality in the following few decades. As such, it will implement a **Risk Assessment framework** supporting the *design*, *test* and *validation* of, not only, a QR TPM, but also a set of mutually interconnected components that will ensure security properties throughout the whole life cycle of cyber physical systems. The goal is to identify a solution that offers effective means to guarantee the specific needs in terms of cybersecurity, privacy and trust, and remain secure not only today, but also in the long term against attacks carried out by adversaries possessing quantum capabilities.

Risk assessment is a key aspect for the efficient operation of Information and Communications Technology (ICT) deployments. Various standards and good practices exist for the establishment of risk assessment which are used to evaluate the effectiveness of mitigation actions and policies that are associated with a given risk. Within the context of FutureTPM, the developed RA Framework will be tailored to the security requirements of TPM-based systems capable of providing a **risk quantification methodology**, which is model driven, and a **run-time risk assessment and software verification mechanism** (especially, for the TPM Software Stack (TSS)) towards achieving *operational assurance* even against newly identified attacks and exploits. While the investigation will cover a wide palette of threats and an identified segmentation of vulnerabilities for the TPM, it will start with the analysis of the core functionalities (Chapter 4) as have been identified in the context of the envisioned use cases (i.e., *Sealing*, *DAA* and *Key Creation and Storage*).

More specifically, this deliverable will mainly focus on the *Risk Assessment* and the *Security Policy Enforcement* components. As aforementioned, the *Risk Assessment* component quantifies the risk that attackers with a quantum computer can steal information managed by use case demonstrators. The computation of the risk is based on using API calls traced with techniques such as extended Berkeley Packet Filters (eBPFs). The computation of the risk will be conceptually separated in two parts: a **design-time** component, and a **run-time** component, which will be implemented as a client application within the host device, and it will be in charge of updating during run-time, the initial risk assessment provided during design-time. The *Policy Enforcement* component will block unsafe usage of API depending on the provided policies, which will be elaborated to tailor the requirements of the three envisioned Reference Scenarios.

1.1 Scope and Purpose

This deliverable defines and documents the **high-level risk assessment** methodology of the FutureTPM platform. In this context, it starts by providing a detailed research background on threat modelling and risk assessment concepts, the methodology that will be used for the FutureTPM, and how these will be applied to the three envisioned reference scenarios. It then continues with a comprehensive overview of the FutureTPM Risk Assessment framework expanding on the interaction among the integral system components. In respect to this, D4.1 aims to derive a clear overview of the framework's conceptual architecture, which will satisfy the

system requirements that have been captured and introduced in the requirement analysis scheme [2]. To this end, D4.1 will provide a thorough system-level architectural specification that will comprise a high-level overview of the architecture layers and components, as well as, the technology axes that will guide the implementation of the particular layers. As D4.1 will guide the development of the technical components comprising the RA platform, this deliverable also includes an initial overview of the interaction patterns and intercommunication schemes between system components, users and third-party entities. Thus, starting from the mapping of the system requirements to platform components, each component will be further decomposed into high-level functional blocks and supported primitives and interfaces.

Towards this direction, this deliverable also provides a concrete meta-model (see **Figure 2**) regarding the calculation of risks that are related with the operation of the QR TPM. More specifically, in the frame of D4.1 the following concepts, along with their relationships, are going to be formalized: **a) risks** that quantify the possibility of harming an asset, **b) threats** that relate to risks; **c) assets** that may be exploited by some of the threats; **d) attack types** related to the attacking surfaces exposed by the assets; **e) vulnerabilities** along with their exploitability and impact; **f) control elements** that can mitigate the effect of exploitation attempts and **g) attestation properties** bound to the control mechanisms.

The usage of the meta-model is twofold. On one hand, the model will be used during design-time by security experts to provide a cartography of their QR TPM supported/enabled applications and services ecosystem. On the other hand, during run-time, the model will be used to serialize all pieces of information that are required to perform the (re-)calculation of relevant risks that may lead to a (possible) dynamic update of predefined security policies. This deliverable provides the normative specification of a meta-model which will be used by security analysts to capture the cartography of a QR TPM supported environment and the non-normative specification of a multi-step RA methodology that has to be applied prior to the risk quantification. It also provides the approach for integrating multiple levels of risk analysis and dependencies such as safety.

Overall, the main goal is to provide the overview of the risk assessment framework, the key technological axes of the risk assessment and vulnerability analysis and to provide a clear distinction on what the risk assessment is going to analyse regarding the TSS and TPM.

1.2 Relation to other WPs and Deliverables

As a threat modelling and risk assessment deliverable, D4.1 arguably relates (and serves as the basis) to all later WP4 deliverables. The use case descriptions, and the FutureTPM reference architecture, provided in WP1 deliverables will be used as the basis for the specific quantitative metrics to be defined in the risk assessment and management phases. Furthermore, this deliverable directly feeds from the activities conducted within the context of WP3 and, more specifically, taking as input the **normative specification of a threat meta-model** (for the core TPM functionalities) and the **inherent trust assumptions** which will be used by security analysts in order to capture the cartography of assets of a QR TPM supported environment. These serve as the core information to structure the **properties (to be attested) and the security policies (to be enforced)** for meeting the desired security, privacy, functional and non-functional requirements.

In what follows, **Figure 1** depicts the relationships of the deliverable to the other Work Packages (WPs). As already pointed out, the outcome of Deliverable D4.1 is intended to support the risk assessment and vulnerability analysis of later activities in this work package (WP). More concretely, it will provide the narrative basis of the architecture requirements of the Risk Assessment framework. The concrete integral components, and the input and output types to communicate between them will be documented. D4.1 will:

a) provide a non-normative specification of a risk assessment methodology,

b) guide the implementation of all integral system components of the RA framework, namely the **Design-time Risk Assessment**, the **Attestation Toolkit** that provide assurance services during run-time and the **Security Policy Enforcement** mechanisms, and
c) constitute the basis for the integration activities, related to the risk assessment of the envisioned use cases, to be performed in the context of WP6.

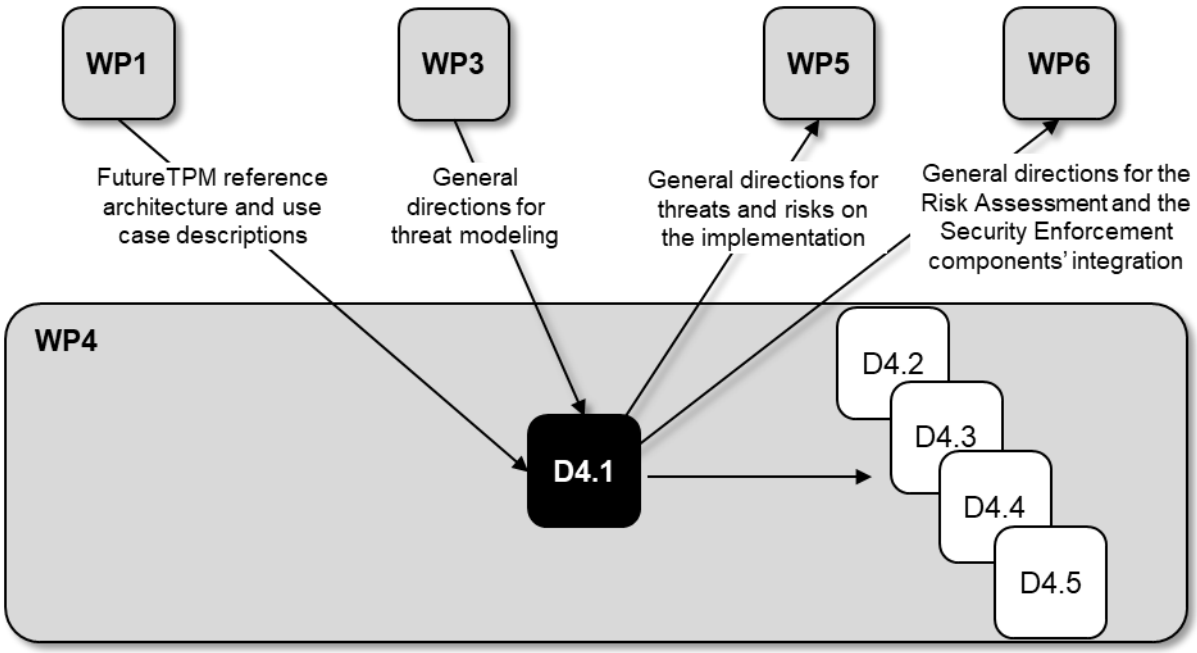

Figure 1: Deliverable D4.1 relationship within the FutureTPM project

1.3 Deliverable Structure

This deliverable is structured as follows. In Chapter 2, we describe and review the research background regarding the definitions, threat modelling and vulnerability analysis and the risk assessment methods; starting by a general overview and then focusing on the specific aspects of TPM-based environments. The overall risk assessment methodology is presented in Chapter 3, where the design-time, run-time and security policy and mitigation enforcement phases are described in detail. More specifically, both risk modelling and risk evaluation as well as the risk modelling toolkit and the quantification engine are described as the main parts of the overall risk assessment framework. Chapter 4 outlines the risk assessment focused on the three envisioned use cases and the possible TPM commands needed per reference scenario. Finally, Chapter 5 concludes the deliverable.

Chapter 2 Research Background

This chapter is devoted to discuss the research background of the state-of-the-art risk assessment methodologies and concepts. In the first subsection, we will focus on an overall **risk assessment approach on cyber physical systems** considering all the assets and their communication, while on the second subsection we will focus specifically on the **threat modelling and vulnerability analysis of TPM and TSS**. Threat modelling will also be investigated in parallel with WP3.

2.1 Definitions and Risk Assessment Methodologies & Tools for Cyber Physical Systems

In this subsection, we consider a number of foundational aspects of risk and vulnerability assessment for cyber physical systems (CPS). CPS is a combination of computation and networking systems with the physical systems that interact together in complex ways and rising uncertainty. We cannot simply apply information security risk assessment. The complex coupling between *cyber* and *physical* components (the TPM in our case) makes it unrealistic to estimate what will happen in the first place. Firstly, we have to define what will happen to the system. Then, we evaluate the probability of the event. At last, we estimate the consequences. During the security analysis of an ecosystem, it is highly crucial to have a clear view of the concepts that are used during the analysis. In order to come up to a common model that will be used during the lifecycle of the entire project, we will adopt concepts from the domain of Risk Assessment (RA). Below we summarize in a consistent vocabulary terms used in this deliverable and will be used in future technical deliverables:

- **Asset**: An asset represents anything that is deemed to be of value. Generally, an asset may be any physical or virtual entity that needs to be protected, such as personnel (employees or customers), material, information (e.g. databases or critical records), or intangibles (reputation or intellectual property). Specifically, within the FutureTPM project, the TPM is the primary asset.
- **Threat**: A threat is an action, or inaction, likely to cause damage, harm or loss. In the FutureTPM context, threats related to the TPM will also be considered.
- **Vulnerability**: Vulnerabilities refer to weaknesses or gaps in the protection of assets that can be exploited via a threat in order to compromise or harm the asset.
- **Risk**: Risk refers to the potential for compromise, loss, injury or other adverse consequence. Risk comes from a threat, or combination of threats, that exploit vulnerabilities in order to compromise an asset. In the FutureTPM context, the primary risk is clearly the risk of a TPM takeover.
- **Control**: As a control we refer to any countermeasure that is applied in order to reduce risk. Practically, the FutureTPM framework is a set of control mechanisms that prevent or minimize specific risks.
- **Impact**: Impact refers to the severity of the consequences stemming from a successful exploit of a specific asset. The impact may be low, medium, high, or even critical in the extreme case of the complete compromise or destruction of an asset. In the absence of a de facto standard for categorization, the FutureTPM-related impacts will be categorized implicitly based on the security characteristics defined in ISO/IEC 25010:2011 [3].

There are probably as many definitions for the above terms as there are formulas to quantify the risk. However, there is one fairly simple equation that establishes the risk by examining the nature of the threats, any perceived vulnerabilities, and the impact if the threat should materialize. The proposed equation is:

$$Risk\_ASSET = Threat * Vulnerability * Impact$$

This is inspired by NIST's guide for security risk assessment [4] according to which the size of a risk is defined per asset based on a formula that quantifies the impact of the associated threats and the vulnerability level for this asset. **Figure 2** provides a high-level view (meta-model) of concepts that are widely used in the Cyber Risk Assessment domain.

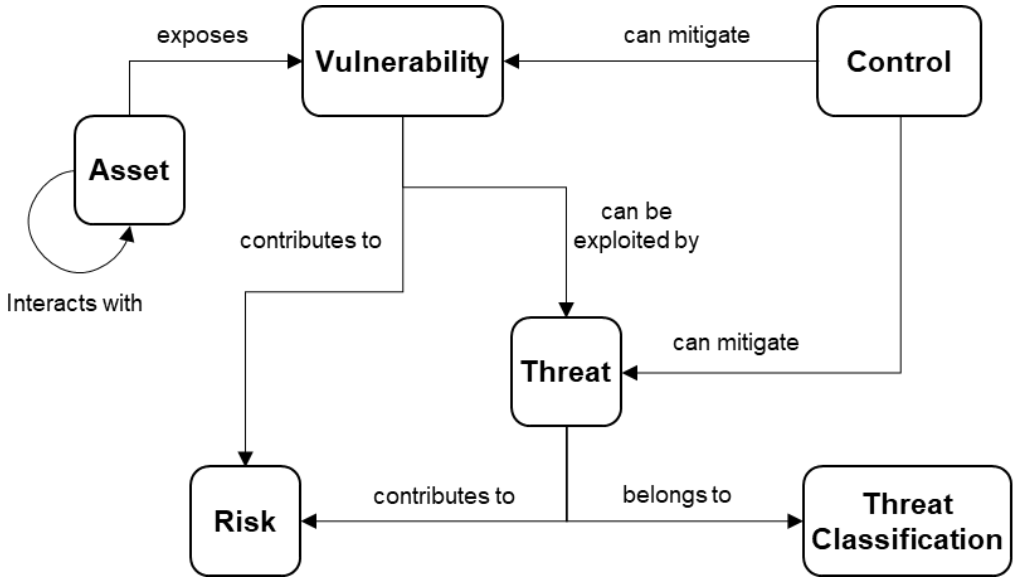

Figure 2: Concepts of Cyber Risk Assessment

The starting point is "Asset". An asset can be a cyber, physical or cyber-physical element within an organization that is used in the frame of business operations. Furthermore, an asset can be something tangible or intangible. Furthermore, an asset may entail a specific business value by itself. Irrelevant of the nature and the type of an asset, each asset may entail several Vulnerabilities. A vulnerability is the stepping stone of the adversary since s/he must identify one in order to harm an Asset. It should be clarified that vulnerabilities are inherent properties of assets and depend on their nature/type. Creating an abstract model for a vulnerability is rather challenging since many parameters have to be taken into consideration. Although there are many models that have been proposed, the Common Vulnerability and Exposure (CVE[1]) model is considered predominant. According to this model a vulnerability is characterized by two properties; exploitability, i.e. how easily you can make use of the vulnerability and impact i.e. how dangerous is the exploitation of this vulnerability. The model defines sub-scores for Confidentiality, Integrity and Availability (a.k.a. CIA) consequences. An open repository for disclosed vulnerabilities can be found here: https://www.cvedetails.com. As it is inferred, a vulnerability may be exploited during an 'Attack' that is able to make use of this vulnerability. In the cyber security domain, the terms Attack and Threat coincide. In the frame of this document and in the scope of the entire project these terms will be semantically equivalent. Analogous to the Vulnerabilities, Threats maintain their own meta-model. A widely used model/taxonomy is Common Attack Pattern Enumeration and Classification (CAPEC[2]). The objective of the CAPEC effort is to describe most common attack patterns classified in an intuitive manner. The attacks were defined using the CAPEC taxonomy which is defined and described below. As seen in **Table 1** below, several types of threats are identified and categorized per domain and mechanism.

[1] http://cve.mitre.org/
[2] https://capec.mitre.org

Table 1: Indicative CAPEC Attack Tree Classification

Category	Attack
Excavation – 116[3]	JSON Hijacking (aka JavaScript Hijacking) - 111
	Directory Indexing -127
	Common Resource Location Exploration - 150
	Cross-Domain Search Timing - 462
	Generic Cross-Browser Cross-Domain Theft - 468
	Probe Application Error Reporting - 54
	Probe Application Queries - 545
	Probe Application Memory - 546
Interception – 117	Sniffing Network Traffic - 158
	Accessing/Intercepting/Modifying HTTP Cookies - 31
	Harvesting Usernames or User IDs via Application API Event Monitoring - 383
	Intent Intercept - 499
Footprinting – 169	Host Discovery – 292
	Port Scanning – 300
	Network Topology Mapping – 309
	Malware-Directed Internal Reconnaissance - 529
	Owner Footprinting – 577
Fingerprinting – 224	OS Fingerprinting - 311
	Application Fingerprinting - 541
Reverse Engineering – 188	White Box Reverse Engineering – 167
	Black Box Reverse Engineering – 189
Protocol Analysis – 192	Cryptanalysis – 97
Information Elicitation - 410	Pretexting – 407
Inject Unexpected Items - 152	Code Injection – 241
	Parameter Injection – 137
	Local Execution of Code – 549

The successful exploitation of a vulnerability through an attack leads to Impact. In order to prevent this impact, specific Control elements should be installed. However, as depicted in **Figure 2** a control element can be applied at the Vulnerability Level or at the Threat Level. The knowledge of assets, vulnerabilities and applicable threats are the prerequisites that have to be defined in order to quantify Risk. In the literature, there are several information security risk assessment techniques, but none is considering the necessities of the TPM & TSS. Different risk assessment methods may obtain different results. We list below some of the most known methods for completeness reasons [5]:

- OCTAVE [6],
- TARA [7],
- FAIR Framework [8],
- COBIT Framework [9],
- NIST Risk Management Framework [10],
- ISO/IEC 27005 [11]

[3] The number following each CAPEC category or member represents its unique identification number, CAPEC-ID, which enables a fast discovery and retrieval of its description.

2.1.1 OCTAVE

The **Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) framework** was created by the Software Engineering Institute (SEI) at Carnegie Mellon University to help organizations perform information security risk assessments. Since its release, different versions were released, OCTAVE Framework v 2.0, OCTAVE Criteria, OCTAVE-S and OCTAVE Allegro. We will focus on OCTAVE Allegro because it differs from previous OCTAVE approaches by focusing primarily on information assets in the context of how they are used, where they are stored, transported, and processed, and how they are exposed to threats, vulnerabilities, and disruptions as a result. There are four activities carried out by eight steps in OCTAVE Allegro methodology:

1. **Establish Risk Measurement Criteria**
2. **Develop an Information Asset Profile**
3. **Identify Information Asset Containers**
4. **Identify Areas of Concern**
5. **Identify Threat Scenarios**
6. **Identify Risks**
7. **Analyse Risks**
8. **Select Mitigation Approach**

2.1.2 TARA Framework

Threat Assessment & Remediation Analysis (TARA) is an engineering methodology to identify, prioritize, and respond to cyber threats through the application of countermeasures that reduce susceptibility to attacks. It is part of the MITRE Mission Assurance Engineering (MAE) portfolio. The MAE portfolio is comprised of an evolving collection of Enterprise Systems Engineering (ESE) practices that combine practical experience, information sharing, research, and experimentation to help sponsors better address the Advanced Persistent Threats (APT). The objectives of a TARA assessment are to identify and prioritize high-risk adversarial Tactics, Techniques, and Procedures (TTPs) that an asset may be susceptible to, to identify and prioritize countermeasures (CMs) effective against those TTPs and to recommend CMs that can reduce the susceptibility of an asset to attack.

A TARA assessment is comprised of two analysis steps:

1. Cyber Threat Susceptibility Assessment (CTSA)
2. Cyber Risk Remediation Analysis (CRRA)

The CTSA step identifies and evaluates the susceptibility of an asset to attack relative to a set of TTPs, while the CRRA step identifies a set of countermeasures that reduce the susceptibility or lessen the effects of an attack.

CTSA consists of the following steps:

1. **Establish assessment scope**
2. **Identify candidate TTP**
3. **Eliminate implausible TTPs**
4. **Apply scoring model**
5. **Construct the threat matrix**

CRRA is performed separately for each cyber asset and consists of the following steps:

1. **Select which TTPs to mitigate**
2. **Identify plausible countermeasures**
3. **Assess countermeasure merit**
4. **Identify an optimal CM solution**
5. **Prepare recommendations**

2.1.3 FAIR Framework

Factor analysis of information risk (FAIR) is a taxonomy of the factors that contribute to risk and how they affect each other. It is primarily concerned with establishing accurate probabilities for the frequency and magnitude of data loss events.

FAIR is also a risk management framework developed by Jack A. Jones now part of The Open group, it can help organizations understand, analyse, and measure information risk. FAIR seeks to provide a foundation and framework for performing risk analyses. Much of the FAIR framework can be used to strengthen, rather than replace, existing risk analysis processes like OCTAVE.

The FAIR framework is based on four primary components: threats, assets, the organization itself and the external environment. It can be used to perform qualitative or quantitative analysis.

The analysis comprises of 10 steps in 4 stages:

1. **Identify scenario components**
   a. Identify the asset at risk
   b. Identify the threat community under consideration

2. **Evaluate Loss Event Frequency (LEF)**
   a. Estimate the probable Threat Event Frequency (TEF)
   b. Estimate the Threat Capability
   c. Estimate Control Strength
   d. Derive Vulnerability
   e. Derive Loss Event Frequency (LEF)

3. **Evaluate Probable Loss Magnitude (PLM)**
   a. Estimate worst-case loss
   b. Estimate probable loss

4. **Derive and articulate Risk**
   a. Derive and articulate Risk

2.1.4 COBIT Framework

COBIT (Control Objectives for Information and Related Technologies) is a good-practice framework by ISACA for IT management and governance. The framework defines a set of generic processes for the management of IT, with each process defined together with process inputs and outputs, key process-activities, process objectives, performance measures and an elementary maturity model. An add-on for COBIT 5 related to information security was released in December 2012. The framework is business oriented and its version for information security is intended for:

- Reducing complexity and increasing cost-effectiveness
- Increase user satisfaction with information security arrangements and outcomes
- Improve integration of information security
- Inform risk decisions and risk awareness
- Reduce information security incidents
- Enhance support for innovation and competitiveness

2.1.5 NIST Risk Management Framework

NIST Risk Management Framework (RFM) tries to integrate security and risk management activities into the system development life cycle (SDLC). It provides emphasis on the selection,

implementation, assessment, and monitoring of security controls, and the authorization of information systems.

It is divided in 6 steps:

1. **Categorize:** categorize the system and the information processed, stored, and transmitted by that system based on an impact analysis
2. **Select:** select an initial set of baseline security controls for the system based on the security categorization; tailoring and supplementing the security control baseline as needed based on organization assessment of risk and local conditions
3. **Implement:** implement the security controls and document how the controls are deployed within the system and environment of operation
4. **Assess:** assess the security controls using appropriate procedures to determine the extent to which the controls are implemented correctly, operating as intended, and producing the desired outcome with respect to meeting the security requirements for the system
5. **Authorize:** authorize system operation based upon a determination of the risk to organizational operations and assets, individuals, other organizations and the Nation resulting from the operation of the system and the decision that this risk is acceptable
6. **Monitor:** monitor and assess selected security controls in the system on an ongoing basis including assessing security control effectiveness, documenting changes to the system or environment of operation, conducting security impact analyses of the associated changes, and reporting the security state of the system to appropriate organizational officials

Each step is supported with NIST special publications, like SP 800-30 that describe how to conduct the risk assessment process.

2.1.6 ISO/IEC 27005

ISO/IEC 27005 [11] is an international standard part of the ISO/IEC 27000-series comprises information security standards published jointly by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). The standard provides best practice recommendations on the management of information risks through information security controls within the context of an overall Information security management system (ISMS).

The ISMS is conducted in 4 phases:

1. **Plan**:
    a. Establishing the context
    b. Risk assessment
    c. Developing risk treatment plan
    d. Risk acceptance
2. **Do**: implement the risk treatment plan
3. **Check:** continual monitoring and reviewing of risks
4. **Act:** maintain and improve the Information Security Risk Management Process

The ISO/IEC 27005 process can be an iterative process.

2.2 Threat Modelling and Vulnerability Analysis

Threat modelling is the process of the systematic enumeration of threats to a system. Just claiming that a system is secure is not enough, we need a method for analysing a system to its components in order to identify all possible threats that it might face. There is no common security solution for all systems, each one has specific security requirements and possible threats and it is required that the enforced policies are uniquely tailored for the system in hand. When choosing these policies, the author must have a holistic view of the entire system with all

the details such as the assets that the system has, all the data flows, the access points, any privileged code and the defined trust boundaries. Threat modelling systemizes processes for identifying all the needed pieces of information that a systems security engineer needs for specifying and enforcing security policies.

Threat modelling produces an analytic schematic of the system that identifies all the possible threats, assesses them and rates them according to their possibility of occurrence and the damage they could do. This model helps to develop realistic and meaningful security policies that perfectly fit the specified requirements. As evident, threat modelling has a close relationship with the definition of the security requirements of the system and the development of the security mechanisms (see **Figure 3**).

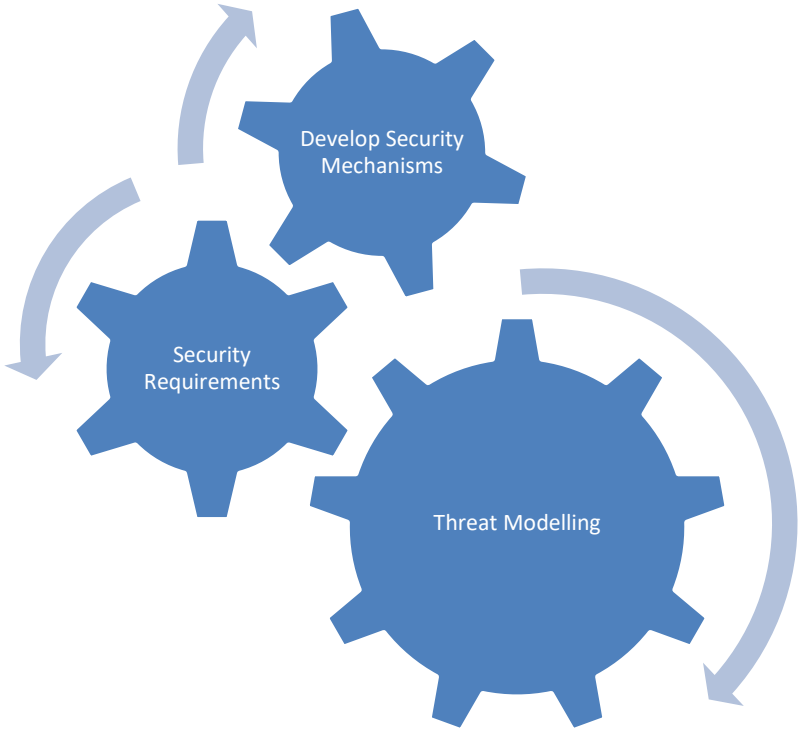

Figure 3: Threat modelling is closely related with security requirements definition and security mechanism development

There are several threat modelling techniques, we are going to evaluate each one and choose the best options that will help to achieve the goals of the FutureTPM project:

- **STRIDE:** The STRIDE technique was developed by Loren Kohnfelder and Praerit Garg in 1999, in a Microsoft technical document as part of the Microsoft Trustworthy Computing Security Development Lifecycle [4]. The name provides a mnemonic for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege, which are the cornerstones of this technique. The STRIDE process consists of decomposing an application into constituent components on data flow diagrams and analyzing each component against possible threats from the aforementioned categories. STRIDE has two different applications, namely on per-element basis [12] [13] and on per-interaction basis [12]. More specifically, the per-element method, focuses on each element that the model has, producing a more in-depth result, while the per-interaction focuses on the interactions between elements. Both methods can be used in conjunction to produce more complete models [14]. The STRIDE method can be used for assessing the threats against the FutureTPM assets, as its focus is around the assets and their interactions.

- **Attack Tree:** Attack trees can be used in the FutureTPM project to model the possible attack paths that lead to specific attack goals. They use tree-like diagrams to represent the possible attack paths that are available in the system. More specifically, the root of each tree is a vulnerability identified to be a threat to the system, the leaves represent the attack surface and the rest of the nodes are subtasks that lead up to the realization of the attack. The modelling process with attack trees has multiple steps [15]:
    - Identify the threats that the system might have and set them as attack tree roots.
    - Decompose each threat to subtasks iteratively until the subtasks consist of basic functionalities which belong to the attack surface. Some tasks might be composed of all of its subtasks, and some only of one, depending on this setting, there are logical AND and OR nodes to be used.
    - Each task is assigned a cost number. If the final attack has a smaller damage than its summed subtasks costs, then the attack is deemed not likely to occur. On the other hand, attacks with a low-cost implementation and highly damaging results, indicate that the attack must be mitigated as it is likely to occur.
- **Attack Library:** Attack libraries are collections of known attacks compiled into searchable databases. They are general purpose checklists, that aim to provide detailed information for attacks, in order to help threat modelers to understand each threat from the perspective of the attacker. Attack libraries can be used in the FutureTPM project to check for known vulnerabilities against the implemented functionalities and cross them out from the list. Some major libraries are:
    - OWASP
    - CWE
    - CAPEC
    - WASC Threat Classification
- **T-MAP** [16] -- It is a quantitative threat modelling technique for assessing security risks by calculating the total severity weights of relevant attack paths. This method begins by identifying attack paths on a four-layer model. The layers consist of: the firewall, the commercial of the shelf (COTS) systems, the IT infrastructure and the organization's core values. These layers are described by 22 attributes which are derived from the Common Vulnerability Scoring System (CVSS). After that, weights are applied to each of those attributes depending on their severity, and each attack path can be evaluated based on those weights. The overall threat of the system can be quantified by summing the weights of each attack path, in order to provide a security evaluation of the whole system. Finally, the threat modeler, evaluates all the countermeasures for each threat, depending on their efficiency and cost. This way, optimal countermeasures can be chosen for the system and minimize both costs and possible damages. This method can provide comparative results against similar implementations in order to assess the threats against FutureTPM.

Until now, we have reviewed threat models that could capture architectural threats, implementation threats and hardware and software vulnerabilities. These techniques, though, are not suited to be used when analyzing the security of cryptographic primitives. In order to assess the security of the cryptography that is (or will be) implemented on the TPM, we will need security models that are designed for this subject. There are two approaches when it comes to cryptographic security modelling: **symbolic** and **computational**. Symbolic cryptography defines cryptographic operations as perfect and unbreakable, thus it is suited for proving that a protocol or application is handling properly the cryptographic primitives it contains. On the other hand, computational security modelling, defines cryptography as probabilistic computations, and it focuses mainly on the actual security of the cryptography. With these two methods combined, we can capture both external (misuse) and internal (chance

of failure) security issues of the cryptographic functions that are implemented in our system. Both methods can be used in the FutureTPM project to check the security level of the implementation while also evaluating the security of the primitives to be used.

In the next subsections we are going to present threat models that have been or can be applied to the TSS part of the TPM and a quantum resistant TPM.

2.2.1 TPM & TSS Threat Modelling and Vulnerability Analysis

The TSS is a middleware that provides a multi-level API to applications for accessing the TPM. Through the APIs provided by TSS, operating system and the users' applications can utilize the security functionality provided by TPM. Until now several TSS implementations exists in different languages, such as IBM TSS [17], Intel's TSS [18], Microsoft's TSS [19], Trousers TSS [20], Java TSS [21], and Daonity TSS [22] etc. Products of trusted computing is mainly concerned with risk management, minimizing the risk to corporate and personal assets due to malicious and accidental loss or exposure. However, an integrated risk management is still considered a breakthrough aspect [23]. More specifically, it handles all the data from and to the TPM, which includes marshalling/unmarshalling data, encrypting transactions for cryptographic sessions, parameter checking etc. The TSS consists of three API layers that provide three levels of abstraction, namely the SAPI, the ESAPI and the FAPI. The FAPI is the most feature rich layer, and its purpose is to cover most of the use cases of the TPM as it includes ready to use functions that require very little configuration. It is designed to be simple to use and make the development of applications as easy as possible. The ESAPI is a little more advanced API and is targeted to individuals who seek to have a little better control over what the TPM does. Finally, the SAPI is an interface that requires expert knowledge of the underlying TPM commands and architecture. The functions it contains can be directly mapped to almost every command that the TPM can execute and it allows for fine grained control of the module. This level freedom allows for misuse and errors, that is why it is of great importance to model all the possible threats that may appear.

The TSS is closely related to the TPM as they are closely interconnected with the TSS being the gateway for all inbound and outbound communications of the TPM. We are going to present the threats models on a per functionality categorization for the TPM and TSS that exist in the bibliography. Furthermore, we will focus on a set of core functionalities of the TPM during the threat modelling process, namely: cryptography, storage, key management, sessions and authorization. The security modelling of the whole TPM is too ambitious and that is why we selected these functionalities that cover most of the use cases to some degree. These functionalities where identified to be common across all the use cases that the FutureTPM project will examine, as they are seen in Chapter 4 of this deliverable.

- **Cryptography, Storage and Key Management:** In [24] there is an analysis of the Protected Storage API security of the TPM. With the developed framework, they showed that part of the API cannot be misused to make an unauthorized access to key objects. The framework provides symbolic security guarantees and its scope is very narrow, as it only focuses on a small subset of the TPM's commands. Zhang et al. [25] presented a model that has a broader scope with its focus being on TPM's key management API. They discovered some possible attacks with their model, but they noted that the source of these threats is not a failure of the TPM specification but a problem with some specific policy definitions. They underlined the need for fine-grained security policies, so that the discovered attacks could be mitigated. Finally, in [26], there is a formal game-based model of the TPM's cryptographic support commands. The researchers used CryptoVerif to prove that honestly generated keys can be used to securely encrypt or authenticate messages, even if the adversary can directly interact with the TPM.
- **Session and Authorization:** In the research of Chen and Ryan [27], they modelled the authorization mechanisms of TPM1.2 and showed that a mistreatment of the

authentication data can lead to complete usurpation of the TPM's secure storage in multi-tenant scenarios, even if encrypted sessions are used. They also used ProVerif to prove the symbolic security properties of a modified protocol that is now used as part of TPM2.0's enhanced authorization mechanism. In [26], the researchers have formalized the HMAC Authorization sessions of TPM2.0 using CryptoVerif. They proved that the TPM only executes protected commands when called by a user who possesses the appropriate credentials and that callers that are engaged in protected sessions with the TPM can trust that execution happened within the TPM upon receiving the results. Shao et al. [28] have made a SAPIC model for a subset of TPM2.0 Enhanced Authorization mechanism. They used Tamarin to obtain proofs of symbolic authentication for most modes of authorization, and identify some cases where misuse is possible and needs to be managed by careful usage of the TPM's API.

- **Direct Anonymous Attestation:** The direct anonymous attestation does not fit into the previous category (i.e., Cryptography, Storage and Key management) since it is a case of its own. It was developed specifically for the TPM and it encapsulates standard cryptography as well as novel and somewhat untested methodologies. A game-based model was proposed in [29] but it was difficult to apply in larger systems as well as it focused on the RSA-based implementation of TPM1.2. [30] found in its ProVerif model that attacks were possible where anonymous credentials could be delivered to the wrong TPM. An ECC-based DAA was analysed in [31] that have led to the development of equivalence-based notions of symbolic security that are used to express strong secrecy properties alongside privacy properties. In [32] the DAA API of the TPM2.0 was modelled in ProVerif that resulted to the introduction of forward anonymity property that was not met in the implementation of TPM2.0. Finally, universally composable security models where used in [33] and [34] that allowed for further refinements of the trust models such as the effect of compromised TPMs to uncompromised ones. Moreover [34] proposed protocol changes for the TPM2.0 DAA that improved its overall security with provable forward anonymity.

The threat modelling of the TPM is a joint effort between WP3 and WP4. The outputs of WP3 will be utilized from WP4 towards the implementation of the **risk analysis** and the **vulnerability assessment** processes.

2.2.1.1 TSS Threat and Vulnerability Modelling

The threat modelling we discussed so far, is targeted at both the TPM and the TSS. These two entities are strongly tied together and there is a thin line that separates one from the other. If we were to target the TSS specifically, we first need to abstract the nature of the TSS to the bare minimum. The TSS is a middleware that provides an API abstraction to the commands of the TPM, and as such it should be handled as a common software. All common software threat modelling techniques apply to the TSS, which are divided in two main categories: white-box and black-box. In the white-box scenario, we have access to the source code of the implementation, and the prime target of this technique is to verify the design, i.e. to assess whether the software does what it is supposed to. On the other hand, the black-box approach, is suited for the "attacker's perspective" scenario, where all we have is access to the compiled and running software, and our target is to enumerate all the vulnerabilities that we can find with this in hand. These approaches remind of the penetration testing process, but threat modelling is nothing more than a systematic method for the identification and the assessment of vulnerabilities to a system, while penetration testing is a systematic process for just the identification (only) of the vulnerabilities.

In the case of the TSS, since we have the source code and a working implementation, we can approach both scenarios. With the use of standard threat modelling tools, we are going to pinpoint all the possible threats to the TSS, and with the results of this process we will attempt to detect all its vulnerabilities. The main techniques that we are going to use are code analysis and fuzzing. The white box approach will be comprised of both automatic and manual code

analysis, which are the basic methodologies for vulnerability detection. We are going to pass the code through automatic code analyzers that will produce a number of possible vulnerabilities then, we will assess these results and identify whether they are actual vulnerabilities or not. After this process, we will compare the vulnerabilities we identified with the results of the threat modelling process, and if we find threats that were not covered by the code analyzers, we are going to manually inspect these points of interest to assess if there are any missed vulnerabilities.

There are four basic techniques for code analysis, all of these techniques are used in automatic code review tools, but some of them could also be applied to manual code reviews too:

- **Control Flow Graph** [35]**:** These graphs are the representation of each possible path that an application might take during its execution. More specifically, each node of the graph represents a block of code that does not contain any out of order execution commands (jumps). All the jumps of the code are represented as directed edges, that point to the target address of the command to be executed after the jump. This way, we can fully map a piece of software and better understand its architecture.
- **Data Flow Analysis** [36]**:** This technique uses control flow graphs, in order to better understand the values that a variable might get. It is a data centric technique, that aims to simplify the code by aggregating the entirety of the commands that act on a variable and produce a smaller subset of these commands with the same result. This method is mainly used in the optimization process of compilers, but it also helps in static code analysis, as it helps to better understand what changes are applied to data and pinpoint possible vulnerabilities or bugs.
- **Taint Analysis** [37]**:** This method, once again expands on the previews one (Data Flow Analysis), with the target of finding how untrusted sources of data are propagated to the system. It utilizes the results from the data flow analysis, in order to “taint” the data from untrusted sources and follow the path of this data throughout the system. This way we have a visual representation of how far an attacker can reach in the system and what components he can access. Taint analysis can be used to identify both information disclosure vulnerabilities and tainted flow vulnerabilities, with the latter being the scenario we described, and the former is the scenario where sensitive information can “sink” into outputs that an attacker might have access to, i.e. the reverse process of the aforementioned scenario.
- **Lexical Analysis** [38]**:** This procedure is applied by automated tools, in order to tokenize source files. With the tokenization, it is easier to spot vulnerable patterns and/or commands that should not be used. This method is mainly used by automated tools.

In the case of the black box (or gray) scenario, we do not have access to the source code, (or we have access to it in the gray scenario) and our main focus is to try and identify vulnerabilities just by a running implementation of the software. The basic and most widely adopted method for the black box scenario is fuzzing. Fuzz testing is an effective technique for finding security vulnerabilities in software. Fuzzers have initially been developed for testing protocol implementations on possible security flaws due to improper handling of malicious input. In [39] the authors present a model-based fuzz framework for systematic automated testing of a TCG Trusted Software Stack implementation. This framework is based on black box fuzz testing methods, integrated with target profiling, data modelling and test algorithm etc. They also demonstrate how their model-based fuzz framework can identity several vulnerabilities in the Daonity TSS [22] implementation. However, what makes fuzzers difficult to use is the fact that a fuzzer by design cannot be general–purpose. [40] is another work that applied fuzzing to a TSS implementation with a tool named Flinder that specializes in discovering typical security-related programming bugs. Although this case study was applied to a TSS that is not used anymore (opentc implementation), the Flinder fuzzer could be also used in the fuzzing of the FutureTPM implementation as it offers custom test suite development capabilities, Linux support and it can be used for both black box and white box fuzzing.

Additionally, there is a technique called symbolic execution which can be considered as a form of white box fuzzing [41]. Symbolic execution has garnered a lot of attention in recent years as an effective technique for generating high-coverage test suites and for finding deep errors in complex software applications. This technique tries to find what parts of the code are triggered by specific inputs that are fed into the program as variables. On each branch of the program, the variable input takes many values so as to check for each possible value what branches of the program are taken. Its main aim is to capture various kinds of errors including assertion violations, uncaught exceptions, security vulnerabilities, and memory corruption in an automated way that may have been missed during the development [42]. Symbolic execution is also used in cryptographic protocols automated verification in conjunction with CryptoVerif as seen in [43]. All things considered, symbolic execution can provide a new perspective in dynamic code analysis and allow for a more precise coverage of the program by potentially finding previously unidentified bugs in the FutureTPM implementation.

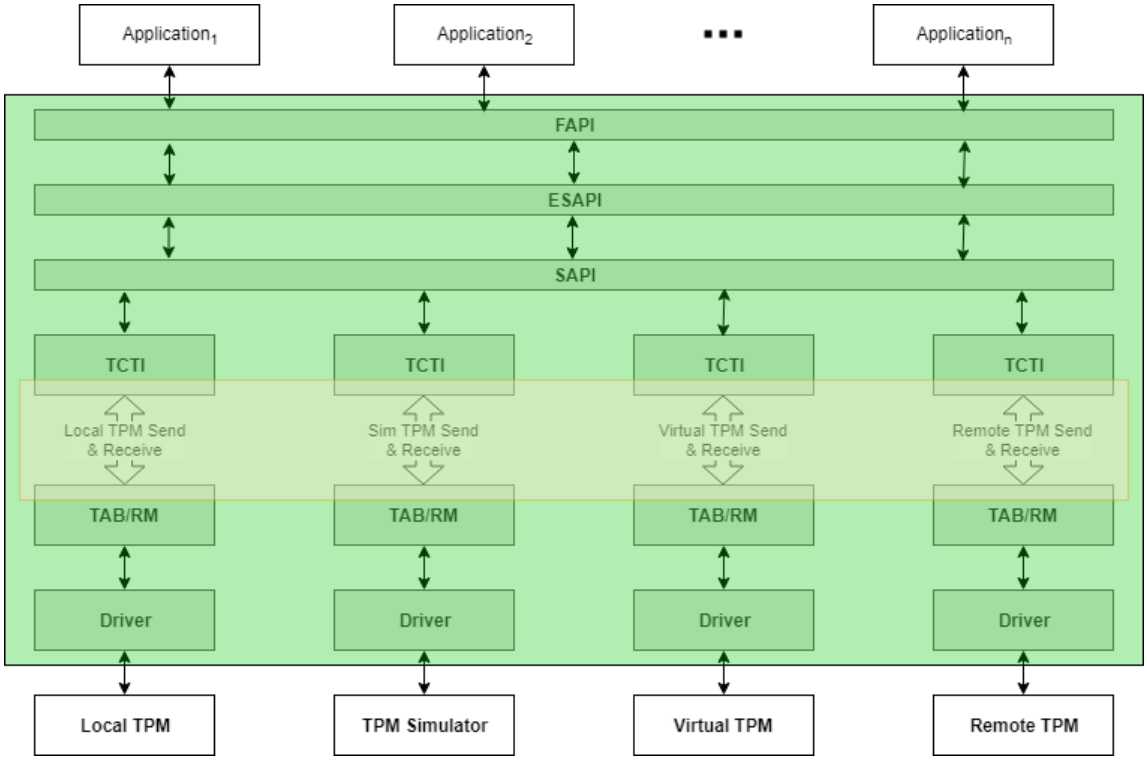

Figure 4: The TCG TSS, the green tinted area will be subject to general purpose fuzzers for testing the TSS, while the yellow area would provide a starting point for fuzzing the TPM.

The architecture of a TCG compliant TSS (see Figure 4), is composed of multiple layers of abstraction. Most of these layers can be accessed by normal or root users and they need to be assessed for the vulnerabilities they might have. On each layer we are going to need specialized fuzzing tools that will cover all possible vulnerabilities. More specifically, general purpose security fuzzers are the best candidates as they cover a wide range of use cases that include the functionality that we need. These fuzzers will be used on all the layers of the TSS, as they cover a broad spectrum of vulnerability detection and they can be configured for most test case scenarios. Furthermore, we will evaluate the need for specialized TPM fuzzers that will increase the effectiveness of the testing process. As of the time of writing, there are no such specialized tools available, it is our purpose to research and develop one, if we find that it is required to reach the FutureTPM goals. The TCTI layer is a special case, as it is responsible for the communication between the functional layers and the lower utility layers of the TSS, that is, it marshals the data to be sent, and forwards them to be handled by the TPM. Because of this, it

provides a good target to test the underlying TPM, and how it handles each packet sent, as we can manipulate these exact packets that the TPM will receive.

2.2.1.2 (QR) TPM Threat Modelling and Vulnerability Analysis

Until now there aren't many research works related to TPM threat and vulnerability modelling let alone for QR TPM, while modelling focused on specific threats in TPM and TPM commands is still missing from the literature. A QR-TPM has both software and hardware components that can provide attack surfaces. There are no solutions in the bibliography that combine both software and hardware threat modelling, with only minor exceptions. A recent paper [44] presented the Lamellae threat modelling framework. In this research, an architectural threat modelling solution is proposed, which combines many modelling techniques in order to cover all the aspects of a system and identify possible multi-level threats. These threats are vulnerabilities that an attacker can build by exploiting the system on different architectural levels. The main absence of this research is the security modelling of cryptography, since it focuses only on threat modelling. In the research of Almohri et al. [45], we can find a similar approach in the threat modelling of medical cyber physical systems. The researchers here proposed a threat model that covers the system on multiple levels (Communication Links, Software, Platform and Users).

Hardware is more resistant to attack than software alone, especially regarding cryptographic key management; this is the strength of TPM. Also, since the TPM uses its own internal firmware and logical circuits for processing instructions, it does not rely upon the OS and is not subject to external software vulnerabilities [46]. Whilst it is widely believed a TPM would reduce risk to an organization, to date there has been no formal risk assessment to confirm it [47]. The threat modelling scheme proposed by Di and Smith [48], focuses on hardware. More specifically, its purpose is to propose a threat modelling methodology that is able to identify possible vulnerabilities that the hardware might have and assess its trustworthiness. The TRUTH tool [49] was proposed in 2008 as a hardware threat modelling tool that uses structural checking [50] attacker-centric checking mechanism. This tool utilizes an RTL (register-transfer level) design of an integrated circuit in order to automatically analyze the possible threats and it also incorporates a method to factor in third party blocks to the overall assessment. In [51] the authors present a risk assessment model based on Bayesian networks. In this model, each risk event influencing the TPM. Bayesian network inferring method was used to evaluate the risk probability and its influence, while the whole system's risk value and risk priority were determined.

Finally, in the research of Ando et al. there is an attempt to improve the quantum resistance of the TPM by embedding a hash-based primitive into its design. The researchers have presented security models for TPM cryptographic functionalities that offload some of the storage to the main memory. These models could be interesting as a basis for our QR TPM setting.

2.2.2 Vulnerability Analysis of QR Algorithms

Now, we will focus on the cryptographic security of the QR FutureTPM algorithms in terms of identifying any possible vulnerabilities. In the work of [52], the researchers have made an effort to identify side channel leakages of an implementation of the lattice-based signature scheme ring-TESLA. They employed both manual code inspection alongside with automated program analysis to find four possible cache side channels vulnerabilities, two of which appear on code that is common across lattice-based implementations. They finally proposed mitigations for the found attack vectors. Furthermore, in [53] the researchers have presented two side-channel differential power analysis vulnerabilities on XMSS(MT) and SPHINCS-256. In the case of XMSS they have successfully recovered all the secret keys of W-OTS+ which allowed them to compromise the security of the scheme. In the case of SPHINCS-256 they managed to reconstruct a 32-bit chunk of the secret key, something that with further research could be extended produce an attack vector that will recover the entire key. Moreover, in the work of

Aysu et al. [54], it has been shown that a new possible side channel vulnerability exists within the matrix/polynomial multiplication of the LWE and R-LWE based key exchange protocols Frodo and NewHope. The main difference between conventional attacks and this one, is that instead of a vertical DPA (Differential Power Analysis), it facilitates a horizontal one that compares traces captured within the same line of operations of a cryptographic function to find leakages.

Adrien et al. [55] have developed an interesting tool for detecting cache-timing vulnerabilities of post-quantum algorithms. The tool inspects the code base of the implementation and makes its judgement based on three factors:

- Secret dependent conditional instruction
- Secret used as index in an array
- Secret used by third-part libraries

Based on these factors, the FutureTPM project can start to identify possible vulnerabilities within the algorithm implementations it will utilize. According to the publishers of the tool, they found that the 80% of NIST submitted QR algorithms contain some potential flaw, with three of them containing more than 1000 reported flaws. The most common flaws between these algorithms that where found with this tool are: gaussian sampling leaks, generic sampling leaks, GMP library usage and operations in finite fields [56]. On the subject of side channel detection, the MicroWalk [57] framework is a useful tool in finding indications of side channels in binaries. It uses dynamic binary instrumentation to locate memory based and control-flow based microarchitectural leakages in binaries and it was used by the researchers to locate possible side channels in closed source implementations of cryptographic algorithms.

Another approach to side channel detection, is to observe the power trace produced during the runtime of the algorithm with different data sets as input. If different data sets, produce statistically different power traces then it is safe to assume that there might be a side channel vulnerability in the implementation. This method is named test vector leak assessment (TLVA) [58] and it utilizes statistical t-tests that can provide a level of assurance that the difference between the power traces is reliable and not a random coincidence. Lei et al. [59] proposed an improvement on this technique that used a frequency spectrum analysis of the power traces. In the conventional TLVA, the power traces had to be aligned in order for the method to be effective, this might not be feasible in some cases, and this new method avoids this constraint by comparing the frequency spectrum analysis results.

In the FutureTPM project we will try to identify and assess possible side-channel attack vectors against the implemented algorithms. Due to the complexity of the task, it is unsure whether there will be any hard evidence for the existence of side-channels. The path that the FutureTPM project will follow, is to first analyze the source code of the algorithm implementation based on the work we described before [55]. Depending on when the final hardware will be available, we will consider to further assess the vulnerabilities of the final hardware implementation using techniques such as TLVA and Differential Power Analysis. With the use of current research endeavors we will enhance our attempt in improving the security of QR cryptography. This work will be part of WP3, WP4 and WP5.

Chapter 3 FutureTPM Risk Assessment (RA) Methodology

This chapter aims at giving a more detailed description of the underpinnings of the risk management phase (documented in D1.2 [1]) covering all aspects of the core services to be offered from **risk identification and quantification to run-time risk assessment and security policy enforcement**. This will serve as the guide for the development of the FutureTPM RA framework to be presented in the subsequent deliverables of WP4.

Risk assessment is a key aspect for the efficient operation of Information and Communications Technology (ICT) deployments. Various standards and good practices exist for the establishment of risk assessment which are used to evaluate the effectiveness of mitigation actions and policies that are associated with a given risk. Within the context of FutureTPM, a risk assessment framework will be developed tailored to the security requirements of TPM-based systems capable of providing a **risk quantification methodology**, which is model driven, and a **run-time risk assessment and software verification mechanism** (especially, for the TPM Software Stack (TSS)) towards achieving *operational assurance* even against newly identified attacks and exploits. While the investigation will cover a wide palette of threats and an identified segmentation of vulnerabilities for the TPM, it will start with the analysis of the core functionalities (more information can be found in Chapter 4) as have been identified in the context of the envisioned use cases (i.e., *sealing*, *DAA* and *key storage creation and storage*).

There is a wide range of generic and TPM specific vulnerabilities that have been identified in the literature, however, what has been lacking is a holistic risk and vulnerability assessment that covers all core components, including QR TPM and the host device, both during *design*- and *run-time*. A work that leverages novel tracing techniques (i.e., use of eBPFs) against a side-channel attack on Java heap management is presented in [60]. Also, [61] presents a run-time kernel attack surface reduction framework restricting the amount of kernel code accessible to an attacker controlling a process, in a quantifiable and a non-by passable way. Both research works will serve as a basis for the FutureTPM risk assessment methodology.

3.1 FutureTPM Risk Assessment Framework

The main part of the Risk Management Phase, is the development of a **generic model to identify threats and vulnerabilities, and a risk assessment methodology for a TPM-based system**. The goal of such a model is to formalize: **a)** the **risks** that quantify the possibility of harming an asset; **b)** all **threats** that relate to specific risks that have been identified in the previous step (this detailed threat modelling is the output of WP3 as (will be) documented in D3.2 and D3.3); **c)** the **assets** that may be exploited in order to manifest some of the threats; **d)** the **attack types** that are related to the attack surfaces exposed by the assets; **e)** the **vulnerabilities** along with their exploitability and impact; f) the **control elements** that can mitigate the effect of exploitation attempts and g) the **attestation properties** that are bound to the control and tracing mechanisms, that were aforementioned (i.e., eBPFs), for ensuring the integrity/verification of the control flow of the executed TPM commands. The high-level overview of the RA framework flow is presented in Figure 5.

As depicted, the core functionality of this framework is the creation and update (in real-time) of the risk graph based on the envisioned application and the types of security properties of interest to be achieved. These properties can be described as security policies that have to be enforced to the deployed cyber-physical system hosting the QR TPM in order to secure the overall platform against the identified risks, or any new vulnerabilities that will be identified during run-time.

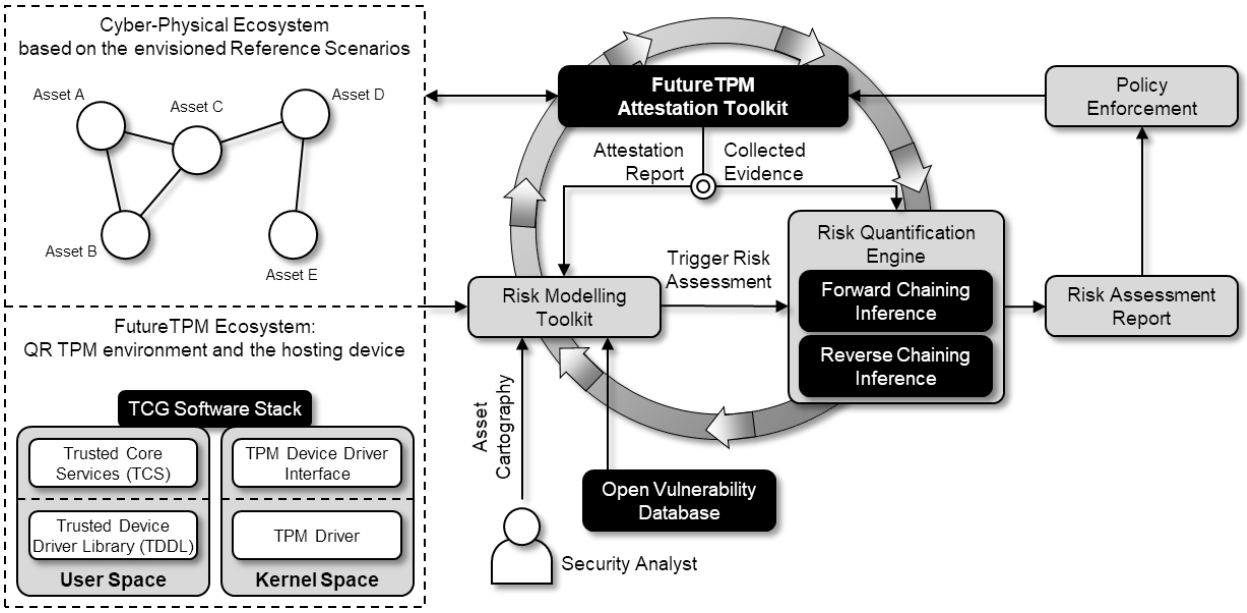

Figure 5: Risk Assessment Framework.

In more details, the four main steps of the Risk Management Phase are as follows:

1. **Design-time Risk Assessment** of the overall FutureTPM architecture towards the creation of asset cartographies coupled with all identified threats and vulnerabilities that can exploited by an adversary. This formalization will allow the definition of adequate security policies and policy sets, capturing all the necessary security requirements (as will be documented in D3.2), that will be made available to the deployed cyber-physical systems which can then start enforcing them and make decisions based upon the results of the run-time risk assessment (including the control flow attestation as part of the **FutureTPM Attestation Toolkit**) processes.
2. **Policy Modelling and Specification** of adequate policies required to mitigate the identified risks. Such policies must be *expressive*, *deployable* and *enforceable* within the **Policy Enforcement Toolkit** and may be dynamically updated if the attack graph (produced and maintained by the RA framework) is amended with new types of vulnerabilities. Example of such policies may include specific properties that will need to be attested during the execution of a system.
3. **Run-time Risk Assessment** on the host device interacting with the (QR) TPM through the TSS. The goal is to provide a real-time calculation of all the risks that have been identified using the **Risk Quantification Engine**; yet the calculation will not only take into consideration a design-time model, that has been produced, but the verified (through attestation) configuration of the deployed platforms. More specifically, through the detailed tracing (based on the use of eBPFs) of the TSS including all the TPM commands and interfaces, kernel shared libraries, etc., an in-depth investigation of the system's behavior and execution flow will be performed to detect any cheating attempts or if any type of (non-previously identified) exploit is resident to the program and data memory.
4. **Dynamic update of the Policy Models** based on any collected evidence regarding the execution behavior of the system; in case of any identified run-time vulnerabilities and exploits and/or data leakage, the RA framework allows the **dynamic change of already established policies**. Run-time security policies may need to be dynamically refined as a response to newly identified attacks that were not evaluated during the Design-time Risk Assessment phase.

In addition to the aforementioned services, as already pointed out in D1.2 [1], the FutureTPM architecture (D1.2 – Figure 7) consists of three main components: the *QR TPM*, the *Risk Assessment* and the *Policy Enforcement* components. The overall architecture of the Risk Assessment framework will take into account all the necessary components and interfaces and how they will interact with the QR TPM through the TSS. Indeed, this is where the novelty of FutureTPM RA lies; **the detailed risk and vulnerability analysis of the whole TPM and TSS environment**. Towards this direction, several commodity TSS implementations exist, namely the Intel and IBM TSS implementation instances. While they share many similarities, Intel TSS [16] provides some additional functionalities (especially when it comes to resource management [70]) that have not been full incorporated yet in the IBM TSS [15]. On the other hand, the current IBM TSS version is at a more stable state. Based on these observations and in order to be able to perform a more holistic investigation in the context of FutureTPM, we decided to leverage both instances: *the Intel TSS will be used as the baseline for the risk assessment analysis whereas the IBM TSS will provide the cornerstone for the implementation and demonstration of the QR-based TPM* (WP5).

3.2 Design-time Risk Assessment Phase

As described in the previous section, the goal of the Design-time Risk Assessment phase is the modelling of the processes (i.e., functions) that are performed through the synergy of various assets; either *cyber* or *physical*. Moreover, the modelling toolkit allows the creation of asset cartographies; i.e., the formal representation of the assets and their relationships. In parallel, the toolkit will make use of open databases in order to associate the modelled assets with existing vulnerabilities (e.g., Common Vulnerabilities & Exposure – CVE Database). Finally, the detailed *sampling* and *tracing* of all the processes will be performed that allows the in-depth investigation of the execution flow of all performed functionalities so as to document a baseline of correct executional behaviour against which the attestation during run-time will then be performed.

Towards this direction, the overall architecture of the Risk Assessment platform will take into account the following functionalities provided by the QR TPM and operated through the TSS:

a) the Components and Interfaces,
b) the Commands and Data Communication Architecture,
c) the Entities,
d) the Hierarchies and
e) the PCRs.

As aforementioned, in order to achieve all the activities of the Risk Assessment, tracing techniques will be employed coupled together with the developed Control Flow Attestation mechanisms (Section 3.3). In addition, both the design-time risk assessment and the security modelling of WP3 will give a first input towards the creation of the necessary policies to be enforced. These policies can then be updated and re-enforced during run-time.

Operating systems and applications are crucial parts of a computer system, but due to their inherent complexity, there are situations related to bugs, incorrect system setup that lead to incorrect behaviour. Compunding these issues, performance statistics collection and their analysis, debug or system audit can help to the system administrator to perform system instrumentation. Two common approaches to instrumentation are [62]:

- **sampling** - when you collect state of the system: values of some variables, stacks of threads, etc. at unspecified moments of time and
- **tracing** - when you install probes at specific places of software. Profiling is a most famous example of tracing

Sampling is very helpful when is unknown what happened. With tracing a probe can be installed to that function(s) of interest, gather information on lists and collect cumulative

execution of function, and then cross-reference them, searching for a pattern in lists whose processing costs too much CPU time [62]. Tracing is used for statistics collection and performance analysis, dynamic kernel or application debug, system audit. Tracing is very powerful but it can also be cumbersome for whole system analysis due to the volume of trace information generated. Unlike other approaches, dynamic tracing tools embed tracing code into working user program or kernel, without need of recompilation or reboot. Since any processor instruction may be patched, it can virtually access any information you need at any place. Dynamic tracing system logic is quite simple: you create a script in C-like language which is translated to a probe code by a compiler. Modern kernel versions have Extended Berkeley Packet Filter (eBPF) integrated, and there is experimentation on using it as a platform for generating probe code.

eBPF has been evolving since 2013 and is a Linux feature which allows safe and efficient monitoring of kernel functions. This has dramatic implications for security monitoring. More specifically, eBPF extends BPF and redefines and extends the set of instructions, relying on common subset from several assembly languages. It is like BPF but with more resources, such as 10 registers and 1-8 byte load/store instructions. eBPF introduces new elements in the architecture, for example the introduction of global data store called maps, whose state persists between events. eBPF gives us a lot of functions called helpers that helps us during the implementation of a BPF program. Therefore, eBPF can also be used for aggregating statistics of events. Further, an eBPF program can be written in C-like functions, which can be compiled using a GNU Compiler Collection (GCC)/LLVM compiler. eBPF can map some memory space so that it can be shared between user and kernel space. It also makes it possible to call certain kernel functions from programs. eBPF programs are loaded from user space but will run in kernel space. The Just-in-time compilation (JIT) is preserved. eBPF has been designed to be JIT'ed with one-to-one mapping, so it can generate very optimized code that performs as fast as the natively compiled code. JIT compiler translates eBPF bytecode into a host system's assembly code and speed up program execution. For instance, on x86 systems, the code is turned by the user space compiler into some "simplified x86 assembly", which is in turn verified in the kernel, and then each "simplified" instruction is translated into real x86 by the JIT compiler. The process is efficient, as:

- all registers map one-to-one,
- most of instructions map one-to-one,
- BPF call instruction maps to x86 call.

These powerful features make eBPF suitable not only for packet filtering, but also for general networking, event tracing, kernel optimizations, and also for the Risk Assessment of the TSS. For clarity, these are some types of eBPF programs (it depends how it is attached to the hook point):

- The **TC (Traffic Control)** programs can be attached to the traffic control layer of the Linux networking stack, both in ingress and in egress. These programs can perform packet manipulation, redirect, clone and in general implement a network function.
- The **XDP (eXpress Data Path)** that provides high performance in the networking (new). XDP programs are attached at the earliest networking driver stage and trigger a run of BPF program upon the packet reception.
- The **Socket Filters** programs can be attached to a socket and they can perform the classic packet filtering process: inject the filter program into the kernel and then the program can return the filtered packet to the user space, and drop the other packets; it was the original tcpdump use case.
- The **Tracing** the idea behind this kind of program is to trigger the eBPF program every time a kernel or system wide event is intercepted. A program can be loaded, and then attached to some events: kprobe, tracepoints, uprobes, USDT probes.

The IO Visor Project [63] is an open source project and a community of developers to accelerate the innovation, development, and sharing of new IO and networking functions. With IO Visor, you can write in-kernel programs that implement atomic networking, security, tracing

or any generic IO function. You can also attach these programs to sockets, so that they'll be executed as traffic arrives in the kernel. The IO Visor Project lists eBPF features available by kernel version (see **Figure 6**). eBPF is not (yet) backed by a rich ecosystem of tools and libraries aimed at simplifying the life of potential developers; the BPF Compiler Collection (BCC) [64] is a toolkit to make eBPF programs easier to write, with front-ends in Python and Lua. BCC project reduces the difficulty for writing, compiling (invokes LLVM/Clang) and loading eBPF programs.

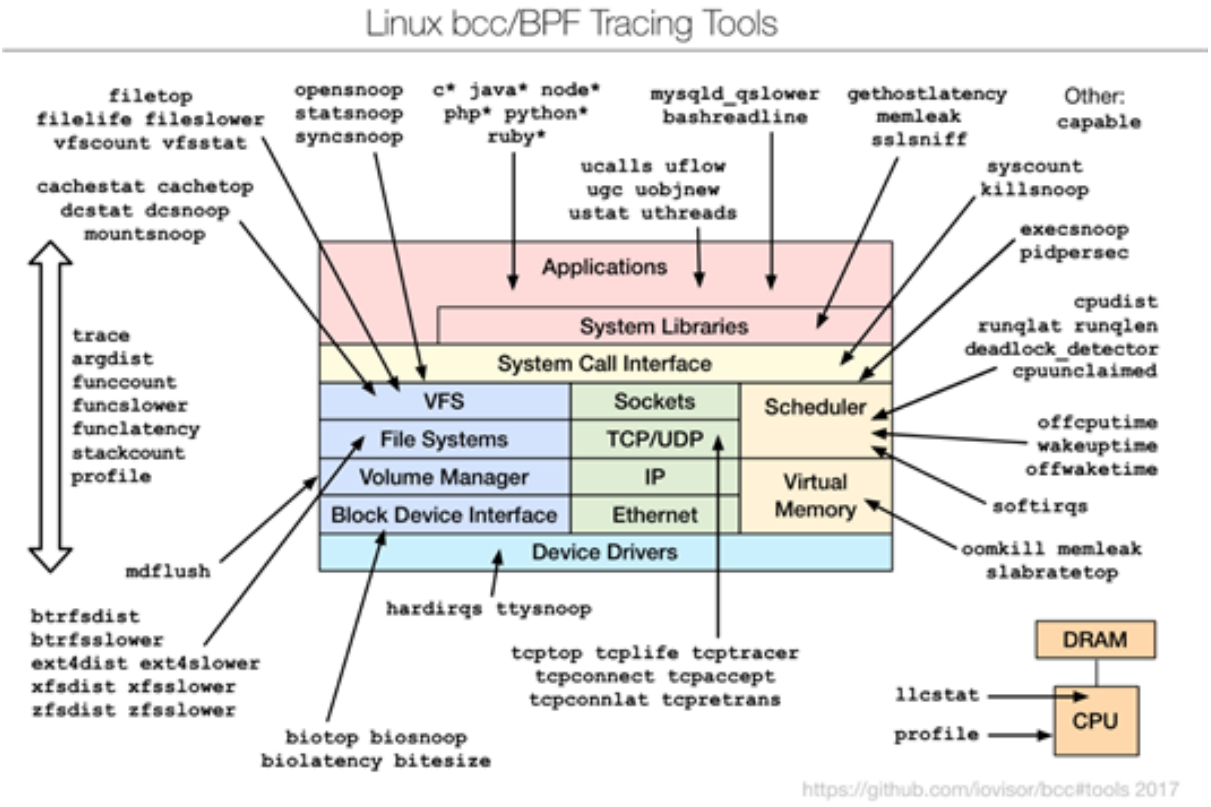

Figure 6: Tracing options [63].

In D1.2, we provided an introduction regarding the **TPM Access Broker (TAB)** and the **Resource Manager (RM)** – two components that are of particular interest due to their inherent functionalities that may lead to sensitive data leakage (e.g., information about stored keys). The TAB controls multi-process synchronization to the TPM. Basically, it allows multiple processes to access the TPM without stomping on each other, while the RM acts in a manner similar to the virtual memory manager in an OS due to limited on-board memory [65]. TPMs generally have very limited memory and objects, sessions, and sequences need to be swapped from the TPM to and from memory to allow TPM commands to execute. RM must parse the command byte stream before the command is sent to the TPM and take any actions required to ensure that all transient objects used by that command are loaded into the TPM. This includes all sessions referenced in the authorization area and all objects, sessions, and sequences whose handles are in the command's handle area. Very recently, the Linux kernel 4.12 have included in-kernel RM [66] to provide isolation between objects & sessions created by different connections which is the core functionality required by applications. Eventually, all of the required features will end up in the kernel RM and it will become the default [66].

FutureTPM will perform a thorough vulnerability analysis of all identified threats and risks that can affect the final product. More specifically, there will be a security analysis of the various TPM environments to ensure that the implementation does not undermine the overall security goals of the FutureTPM platform. A low-level interception at the TPM host device to identify and provide erroneous usage of the TPM that considered QR broken, without interrupting the TPM

is necessary. In D1.2 the cartography of broken commands is provided and can serve as a basis for the implementation of a transparent tracer who intercept these commands (along with the specific reference scenarios TPM commands), based on the eBPF. More specifically, we need to trace the aforementioned TAB and RM between the user land and the kernel (Section 3.3.3). The main Risk Assessment components are the *Risk Modelling Toolkit* and the *Risk Quantification Engine*. The following subsections describe the risk modelling and risk evaluation phases and how the *Risk Modelling toolkit* and the *Risk Quantification Engine* intercommunicate between them.

3.2.1 Risk Modelling

All WP3 deliverables will serve as a reference point when it comes to the subsequent risk modelling. More specifically, the risk modelling will get the necessary input from the threat modelling and trust definitions that will be conducted in the context of WP3. In terms of the FutureTPM Risk Assessment, the risk model output will formalize:

- **risks** that quantify the possibility of harming an asset;
- all **threats** that relate to specific risks that have been identified during the design time;
- **assets** that may be exploited in order to manifest some of the threats;
- **attack types** that are related to the attacking surfaces that are exposed by the assets;
- **vulnerabilities** along with their exploitability and impact;
- the **control elements** that can mitigate the effect of exploitation attempts and
- **attestation properties** that are bound to the control mechanisms of the TPM.

To develop a holistic QR TPM-solution, it is crucial to take into consideration the complex threat landscape posed by the ecosystem of the devices hosting the QR TPM. As already pointed out eBPFs will be used to capture the execution of the command flows in the device (hosting the TPM) so that we can check and attest the integrity of the execution behavior based on already defined policies. Failing to adequately tackle this increased surface of attack, may allow an adversary with quantum capabilities to mount successful attacks and recover secret information. Therefore, in addition to a risk assessment methodology during design-time, it is also required to develop a reactive run-time risk assessment (Section 3.3) and mitigation framework (Section 3.4) for the whole TPM-based solution to ensure security of use cases in the face of emerging threats and vulnerabilities.

As depicted in Figure 5 through the usage of the *Risk Modelling toolkit* a security analyst will model several processes that are performed through the synergy of the various services within a TPM environment. Moreover, the *Risk Modelling toolkit* will allow the creation of asset cartographies i.e. the formal representation of the assets and their relationship. In parallel, the toolkit will make use of open databases in order to associate the modeled assets with existing vulnerabilities (e.g., Common Vulnerabilities & Exposure – CVE - database). Finally, the properties that can be potentially attested per each asset will be also provided as an input to the *Risk Modelling toolkit*. Based on the asset cartography, the cyber physical ecosystem envisioned on the use cases and databases of existing vulnerabilities, the design-time risk assessment module is going to produce a risk graph. After the creation of model instances, the security analyst may trigger the Risk *Quantification Engine*.

3.2.2 Risk Evaluation

After the model creation, the security analyst may trigger the **Risk Quantification Engine**. This engine will be multi-threaded (by-design) since each separate risk quantification request requires different set of calculations. One of the FutureTPM consortium members (UBITECH) has implemented and launched the OLISTIC Risk Assessment Platform covering cybersecurity, information security and privacy impact assessment [67]. OLISTC platform can assist to perform dynamically, continuously and near-real-time risk assessment, addressing the various possible

cascading effects that are associated with incidents occurring from interacting entities and assets. In addition, OLISTIC is constantly updated via the US National Vulnerability Database with the latest identified vulnerabilities and exposures (CVE) and the naming scheme for information technology systems, software, and packages (Common Platform Enumeration - CPE Dictionary). OLISTIC calculates risk levels across assets, asset groups, process, organizational units or other business aspects. This platform will be used to perform the FutureTPM risk evaluation and calculate the overall risk graph. Moreover, the Drools [68] efficient expert system will possibly be used as a cornerstone component for the sake of the engine implementation. *This technical choice implies that several rules have to be created (automatically) regarding the calculation of vulnerabilities, generation of attack trees, propagation of an exploitation (cascading effect), etc.* As any expert system, the developed engine will be able to be used in two modes, *forward-inferencing* and *reverse-inferencing* mode.

3.2.2.1 Forward Chaining Inference

Forward-chaining is a bottom-up computational model. It starts with a set of known facts and applies rules to generate new facts whose premises match the known facts, and continues this process until it reaches a predetermined goal, or until no further facts can be derived whose premises match the known facts. It checks the facts against the query or predetermined goal and indicates that the inference moves forward from the facts toward the goal [69].

In the context of FutureTPM, according to the forward-inferencing mode the engine will produce raw calculation of the risk likelihood for a specific setup of the assets. Threat assessment in most environments consider two metrics: Likelihood of an attack and impact of the attack. Underlying these metrics are a further set of metrics addressing such issues as availability requirements (i.e. time needed to access the vulnerability), equipment (i.e. the complexity or cost of equipment needed to launch the attack) and so forth which are described in some detail in ETS TS 102 165-1 [70]. The calculation of risk is taken most often as the product of likelihood and impact and categorized as high, medium or low, defining countermeasures against high and medium risk vulnerabilities.

3.2.2.2 Reverse Chaining Inference

Backward or reverse chaining it is a top-down computational design and starts with a goal or hypothesis and looks for rules to support the hypothesis. It attempts to match the variables that lead to valid facts in the data and indicates that the inference moves backward from the intended goal to determine facts that would satisfy that goal [69].

In the context of FutureTPM, according to the reverse-inferencing mode, the engine will be provided a specific goal regarding a risk level and then it will propose a set of setups that can satisfy this goal. This setup mainly constitutes the set of properties that if attested can achieve the desired level of assurance. Therefore, the first mode is rule-driven while the second one is goal-driven. The report that is generated, in both cases, can be interpreted in concrete policies. More specifically, the output of the *Risk Quantification Engine* will be assessed in order to evaluate if a set of specific risks are below acceptable thresholds. The thresholds will be set based on the envisaged deployment scenarios driven by three of the consortium leading industrial partners. In case of runtime RA, near real-time risk quantification of newly identified attacks (during runtime) will also be performed. In particular, new target values will be set as input to the *Risk Quantification Engine* which will be invoked in the mode of goal-driven inference engine. This means that it will propose several new setups i.e. mitigation controls that map to existing properties that have to be attested. These properties may be a subset of the configuration properties that are already defined or can be other newly-identified, high-level properties that can further enable semantic remote attestation, i.e., attestation of dynamic, arbitrary and system properties as well as behavior of executable code in an attempt to mitigate the newly discovered runtime vulnerabilities. This process may lead to a dynamic update of the already defined policies as a response to these newly identified attacks.

3.3 Run-time Risk Assessment & Assurance Phase

The scope of this phase includes the provision of **secure, robust and efficient run-time behavioural attestation methods to check the internal state of a remote "untrusted" CPS**, hosting the TPM, towards establishing the **trustworthiness** of the entire "Systems-of-Systems" (SoS). To that end, FutureTPM will build on-top of existing remote attestation techniques (namely, DAA and a combination of Static and Dynamic Attestation [1]) towards ensuring the internal state of a TPM-based CPS in order to enhance its security and privacy posture. These methods will be used for achieving trustworthiness of the command execution flows, as monitored during run-time, based on the tracing capabilities provided by the Design-time RA component and the use of eBPFs, for the detection of any suspicious activities.

3.3.1 Remote Attestation towards System Assurance Enhancement

Remote attestation is a means of integrity verification of software running on a remote device. It is a mechanism typically realized as a *challenge-response* protocol, which enables a trusted party (verifier) to obtain an authentic, accurate, and timely report about the software state of a potentially untrusted service, application or device (prover). The verifier can then check whether the reported state is **trustworthy**, i.e., *whether only known and benign software is loaded on the prover*.

The standard trust assumption needed for authenticity of the attestation report requires the existence of a trusted component that acts as a **trust anchor**. Thus, in the context of FutureTPM, the **CPS itself hosting the TPM can act as the verifier**[4] for attesting the software state of the "*untrusted world*" provided by the host device. The question of interest in this model is whether the TSS that provides the necessary API for accessing the "trusted world" of a TPM, and runs in the untrusted world, can be protected against a wide surface of attacks (or, at least, sufficiently monitored so that any malicious activities can be identified) so that we can attest its state of trustworthiness towards performing the required functionalities and commands. In a more general context, this question can be translated to: *what types of services can be placed in the untrusted world without compromising the overall security and privacy of the system?*

The reason behind employing remote attestation mechanisms as a means of operational assurance is twofold: First of all, one of the main challenges in managing device and network security in today's heterogeneous and scalable infrastructures is the lack of adequate **containment and sufficient trust** when it comes to the behaviour of a remote system that generates and processes mission-critical and/or sensitive data. An inherent property in FutureTPM is the codification of trust among computing entities that potentially are composed of heterogeneous hardware and software components, are geographically and physically widely separated, and are not centrally administered or controlled. By leveraging the artefacts of traditional security infrastructure (such as digital signatures, certificates and assurance statements) coupled with advanced crypto primitives (such as run-time property-based attestation) and building upon emerging trusted computing technologies and concepts, FutureTPM will convey trust evaluations and guarantees for each network entity.

This high level of trustworthiness which will not only include integrity of system hardware and software but also the correctness and integrity of the generated data flows will, in turn, reduce the overall attack vector and allow for the more effective operation of the FutureTPM security framework. This will allow the secure configuration, deployment and operation of distributed, scalable "*Systems-of-Systems*" infrastructures.

Most current remote attestation approaches are static in nature. In such schemes, the prover's report is typically authenticated by means of a cryptographic signature or a MAC

[4] Or it can be that the security analyst acts as the verifier

computed over the verifier's challenge and a *measurement* (typically, a hash) of the binary code to be attested. However, static attestation, though efficient, only ensures integrity of binaries and not of their execution. In particular, it does not capture software attacks that hijack the program's control flow [71]. These attacks tamper with the state information on the application's stack or heap to arbitrarily divert execution flow. State-of-the-art memory corruption attacks take advantage of code-reuse techniques, such as return-oriented programming, that dynamically generate malicious programs based on code snippets (gadgets) of benign code *without* injecting any malicious instructions [72]. As a result, the measurements (hashes) computed over the binaries remain unchanged and the attestation protocol succeeds, even though the prover is no longer trustworthy. These sophisticated exploitation techniques have been shown effective on many processor architectures, such as Intel x86 [73], SPARC [74], ARM [75], and Atmel AVR [76].

The problem arises because static attestation methods do not capture a program's runtime behaviour (i.e., timely trustworthiness) of the underlying code. To be truly effective, an attestation technique should report the prover's dynamic state (i.e., its current execution details to the verifier) so that it can capture control-flow attacks at the binary level of the OS and the SAPI level of the TSS (Figure 4).

Current defences against such control-flow hijacking include control-low integrity [77], fine-grained code randomization [78] and code-pointer integrity [79]. However, simply integrating these approaches into the FutureTPM RA framework would provide limited state information to the verifier. In particular, these techniques only report whether a control-flow attack occurred, and provide no information about the actually executed control-flow path. Therefore, the verifier cannot determine which (of the many possible) paths the prover executed. This limitation allows an attacker to undermine such defences by means of so-called data-oriented exploits [80]. These attacks corrupt data variables to execute a valid, yet unauthorized, control-flow path. Hence, the need for enhanced tracing so that adequate information (evidence) can be collected on the execution instance of the TSS.[5]

3.3.2 FutureTPM Control Flow Attestation Toolkit

Compounding these issues and in order to cope with the ever-increasing attack surface targeting the dynamic execution properties of CPS, FutureTPM RA shall enable the provision of **automated and scalable behavioural-based attestation services** (through the **FutureTPM Attestation Toolkit**), reflecting the identified (and configured during run-time) **preventive, access control, information flow and functional safety policies** that will be enforced by the **Security Policy Enforcement** component (Section 3.4). To this end, the properties (of interest) to be attested by the deployed cyber-physical systems will vary depending on the type of functionality that each such device offers. A generic assurance technique needs to be applied capable of coping with all these different specifications. FutureTPM aims to overcome this challenge by having a general **behavioural-based attestation mechanism developed based on the novel concept of control-flow attestation**.

Control-flow attestation [81] is one of the most important dynamic properties at the software layer since it captures diverse instantiations of software exploits that hijack a program's control flow. Such attacks tamper with state information in the program's data memory area, e.g., the stack and the heap. Software bugs allow an attacker to arbitrarily alter state information and hijack the program flow of applications to induce malicious operations. While traditional attacks require the attacker to inject malicious code [82], state-of-the-art attacks such as return-oriented programming leverage code that is already present in the vulnerable application thereby bypassing modern mitigation strategies [83]. In other words, the attacker resembles malicious codes through a combination of already existing benign code pieces.

[5] To alleviate this problem, eBPFs are employed in the context of FutureTPM RA.

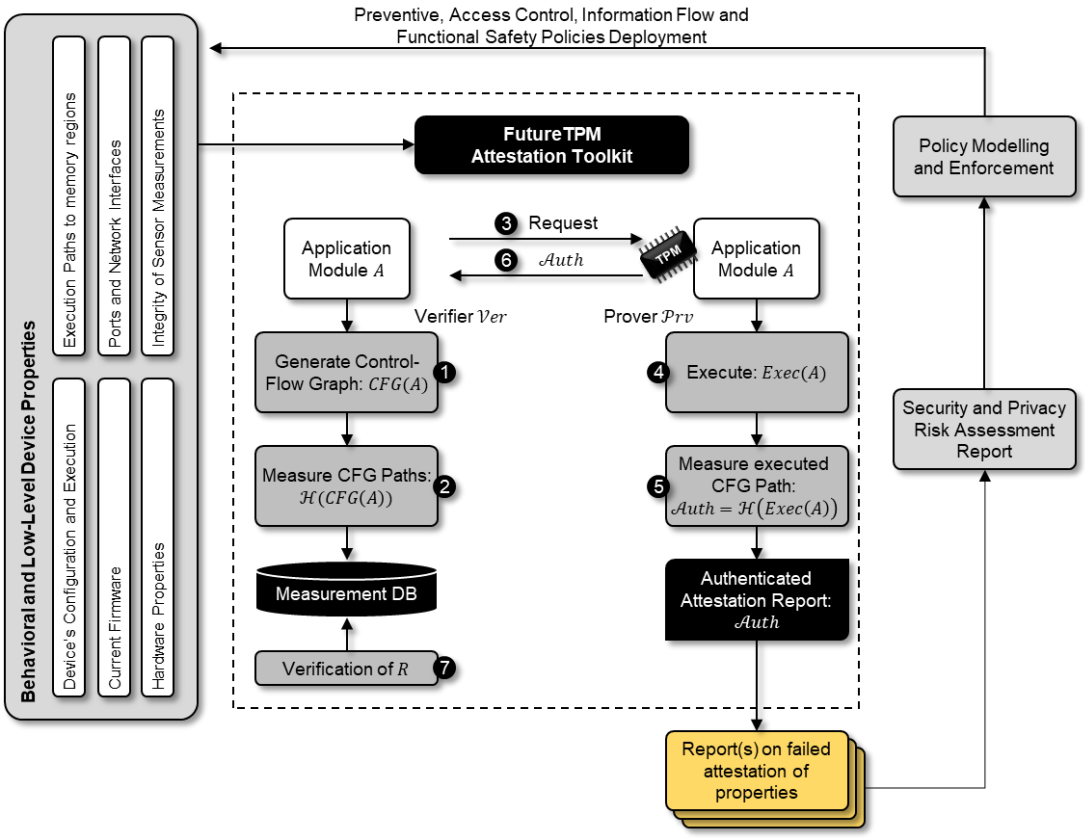

Figure 7: FutureTPM Control Flow Attestation Toolkit

The general approach of control-flow attestation that will be developed as part of the FutureTPM Attestation Toolkit is depicted in Figure 7. It shows a cyber-physical component (hosting a TPM), acting as the verifier, that first receives the necessary security policies containing the specifics of the properties to be attested (see Section 3.3.2.1 for more details). Based on the interpretation of these policies, it then computes all legitimate control-flow paths of a software (i.e., TSS), and store its measurements in a database (**Steps 1 and 2**). To trigger the run-time attestation, as dictated by an already defined security policy, the verifier sends a request to the device which acts as the prover (**Step 3**). The prover device executes the software that the verifier desires to attest (**Step 4**) and the TPM measures the taken control-flow paths (**Step 5**). For instance, this can be achieved through a hash function. Finally, the attestation result is send back to the verifier for validation (**Steps 6 and 7**). In the case of a failed attestation about a system's integrity, the information might not be sufficient to understand the device's behaviour. Thus, in this case, a more in-depth investigation of the system's behaviour is needed to detect any cheating attempts or if any type of (non-previously identified) malware is resident to the system. The goal of this functionality is to then feed this detailed analysis to the Risk Assessment component of the Security Policy Enforcement mechanisms for dynamically defining new attestation policies against this newly identified attack vector (Section 3.4.1).

This conceptual architecture will steer the implementation of the proof-of-concept FutureTPM Attestation Toolkit that will be documented in details in D4.2 for the envisioned use cases. In this context, further investigation will be conducted on operation specifics including: *(i)* the channel on which the attestation results will be reported, *(ii)* the interval and frequency of attestation, *(iii)* the enforcement of the attestation process on the prover's device, *(iv)* group attestation of a SoS, and *(v)* the analysis and categorization of the trust assumptions based on hardware characteristics of the devices hosting the TPM (for each use case). For the latter, these trust assumptions will be documented in the D3.2.

3.3.2.1 Proof-of-Concept Architecture

As aforementioned, in order for the FutureTPM RA framework to perform control flow attestation, the verifier asks for a measurement of the prover's execution path. This measurement should allow the verifier to efficiently determine and verify the prover's control-flow path. For instance, in the context of the TSS attestation, information of interest will include: **execution time, process name of invoked libraries, process id, time of internal operations, parsed TPM commands**, etc. As will be described later in this section, this information is the output of the tracing process performed through the use of eBPFs (Section 3.3.3 gives more details on the already investigated TSS Resource Manager).

However, it is clearly infeasible to record and transmit every execution instruction, since this would: (1) result in a very long attestation response, and (2) require the verifier to walk through every single instruction/command. Compounding these issues and to allow fast verification, FutureTPM will enable **property-based attestation based on the configuration and execution properties of the target device.** The intuition behind this approach is that we are not interested in the attestation of the overall device execution (which is infeasible to achieve efficiently during run-time) but on specific execution properties that constitute the core functionalities of a device featuring specific services. These services are the core TPM functionalities that will be investigated in the context of the envisioned use cases, namely **sealing, DAA and key creation and storage** (Chapter 4).

This is to include both **behavioural properties** and **low-level concrete properties** about the entity's configuration and execution, such as *the current firmware version it is running, the version of its configuration file or presence of certain hardware properties, integrity of sensor measurements, execution paths to specific memory regions, ports and network interfaces, etc.* It also includes abstracting these low-level values to higher level security properties or functions. Should an entity deviate from the current security policy, the FutureTPM RA framework will instruct the entity to collect additional evidence that can be used to either prove its trustworthiness or to determine a route to meeting the security policy. For example, it may request the entity to collect more logging information through more intense tracing. While some static properties may be already defined at design-time, most of them will require runtime verification. Furthermore, while some of the properties can be evaluated for a single entity in the system and can, thus, be verified by a node-to-node attestation, others will require a larger SoS viewpoint and can potentially only be identified on a central verification entity.

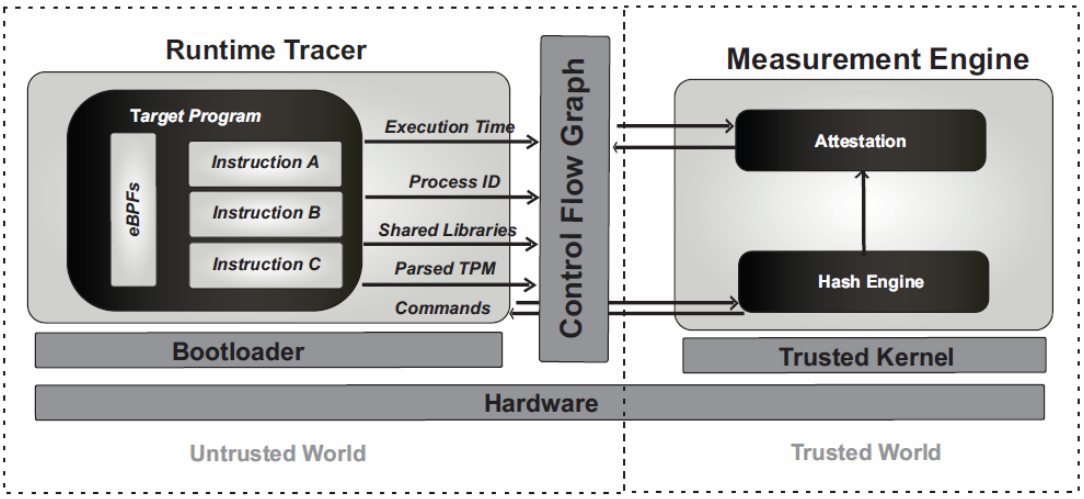

Figure 8: Proof-of-concept Architecture

Figure 8 provides an overview of the proof-of-concept architecture to be implemented within the context of the FutureTPM Run-time RA framework and that will be presented in detail in the context of D4.2. This includes two main components: (1) a **Runtime Tracer** based on the

use of eBPFs and a **trusted Measurement Engine** to trace and measure the run-time control flow path leveraging the output of the execution of the eBPFs. The former can be realized either as: (1) a static binary analyzer that generates the program's CFG by identifying basic blocks and their connection, or (2) a dynamic analyzer that produces valid measurements for a set of inputs by tracing execution of the target program. The eBPF-based tracer needs to be programmed to intercept (as much as possible) all internal operations. One inherent assumption made is that the Measurement Engine needs to be trusted; basically, cannot be the altered or compromised during execution since this will allow an adversary to manipulate the monitoring of the control-flow path execution. This can be achieved through incorporating traditional static attestation techniques of securely measuring and keeping the hash of this engine binary signature (during boot-up) in the TPM and periodically verifying its correctness.

3.3.3 Case Study: TSS Resource Manager (RM) Instrumentation

This section gives an overview of the first instance of the implemented Runtime Tracer and how it operates in the context of trying to extract detailed information for two core components of the TSS stack:

- **TPM Access Broker (TAB),** and
- **Resource Manager (RM)**.

TPMs are very limited in their memory (this is mainly due to reduce their cost). This means objects, sessions, and sequences must be swapped in and out of the TPM as needed, much like a virtual memory manager swaps memory pages to and from disk drives. In both cases, the calling application thinks it has access to many more objects and sessions (in the TPM case) or much more memory (in the virtual memory case) than can actually be present at any given time. For a system where only, a single application sends commands to the TPM, these swapping operations can be performed by the application itself. However, when multiple applications and/or processes are accessing the TPM, two components of the TSS stack are required [84], namely the TAB and the RM.

The TAB and RM were first described in D1.2 [D1.2] as layers in the TSS. A TPM command can use at most three entity handles and three session handles. All of these need to be in TPM memory for the TPM to execute the command. The job of the Resource Manager is to intercept the command byte stream, determine what resources need to be loaded into the TPM, swap out enough room to be able to load the required resources, and load the resources needed. In the case of objects and sequences, because they can have different handles after being reloaded into the TPM, the RM needs to virtualize the handles before returning them to the caller. As already pointed out the TAB and RM transparently isolate TPM applications and processes from the messiness of arbitrating multi process access to the TPM and swapping objects, sessions, and sequences in and out of TPM memory as needed. The TAB and RM are closely related and typically integrated into the same software module. Depending on system design, they may reside in the top layer of a TPM device-driver stack, or they may be integrated as a daemon process sandwiched between the TSS system API layer above and the TPM device driver below. The TAB's responsibility is fairly simple: arbitrate multiple processes accessing the TPM, while the RM is responsible for transparently handling all the details of swapping objects, sessions, and sequences in and out of the TPM.

RM provides isolation between objects & sessions created by different connections which is the core functionality required by applications and very recently included in the Linux kernel. If a transient entity is used in a command, it must be loaded into TPM memory. This means that an RM must parse the command byte stream before the command is sent to the TPM and take any actions required to ensure that all transient objects used by that command are loaded into the TPM. This includes all sessions referenced in the authorization area and all objects, sessions, and sequences whose handles are in the command's handle area. Also, we will think the TPM as a black box. As already pointed out, the Linux kernel 4.12 have included in-kernel RM [66]

and eventually, all the required features will end up in the kernel RM and it will become the default [66].

As described in the previous section, one important aspect of the FutureTPM Attestation Toolkit is the interdependence with the employed eBPF hooks for extracting detailed information on the execution path. This is one of the main goals here: *to hook eBPF in the in-kernel RM, trace all the TPM commands and identify possible object, sequence, session leakage and proven broken TPM commands*. We will focus on the northbound interface of the TSS stack the SAPI. The first release of the framework will be further analysed in D4.2.

In this first release, we will mainly focus on the three use cases and on how to trace the all the needed TPM commands per use case (see Chapter 4). Currently the framework is a work in progress, and an initial tracing example of the TPM2_Create command achieved during run-time. We trace the read (R) and write (W) operations from/to the kernel. Extracted information re provided through appropriate figures in Appendix A, where the TPM first loads the parent key and then creates the new one. We can see the TPM_CC_ContextLoad and TPM_CC_Create command codes send to the TPM with the corresponding TPM_RC_SUCCESS responses. This example is a part of our initial implementation for the FutureTPM Risk Assessment Framework. At this stage, the output provides the following information (see Table 2).

Table 2: FutureTPM Risk Assessment output information

	Output Information

Kernel Space	• the amount of **time passed** since the framework was executed in seconds (e.g. 2193.54)
	• the **process name** (e.g. tpm2_create)
	  ○ Is limited (truncated) to 16 bytes.
	• the **process id** (pid) (e.g. 8094)
	• the **Virtual File System** (VFS) function hooked
	  ○ VFS Open, VFS Read and VFS Write operations (e.g. O, R, W)
	• the **number of bytes** read (e.g. 938 of 938)
	• the **time of the operation** to complete in milliseconds (e.g. 37.89)
	• the **type of files** (e.g. CHAR DEVICE)
	  ○ The Linux kernel considers nearly everything to be a file. That includes directories and devices: They're just special kinds of files such as:
	  ▪ Directory
	  ▪ Symbolic Link
	  ▪ Block Device
	  ▪ Character Device
	  ▪ Named Pipe (FIFO)
	  ▪ Socket
	• the **device / file name** (e.g. tpmrm0)
	  ○ in our case the resource manager
	• the **parsed data** from kernel with eBPF
	  ○ in ascii format and
	  ○ in hex for non ascii characters

User Space	• the **parsed TPM commands** ○ output is based on the TPM specification (e.g. Command Header and Parsed Command)

3.4 Security Policy & Mitigation Enforcement Phase

The **Security Policy Enforcement (PE**) mechanism will utilize the results from the risk assessment process in order to provide a re-active policy enforcement methodology. Defined policies must be *expressive*, *deployable* and *enforceable* within the PE tool and may be dynamically updated if the attack graph (produced and maintained within the Risk Management phase) is amended with new types of vulnerabilities. As described in the previous section, FutureTPM's Run-time Risk Assessment and Assurance toolkit must also address the dynamic run-time properties of all deployed CPSs which cannot be modelled completely at design-time; taking into consideration their changing control objectives and system configurations as well as the changing communication and computation resources and allocation. Such run-time system models are also enhanced with hierarchical control layers providing different levels of assurance for various types of behavioural properties.

The enforcement process will take place in both the design- and run-time phases of the development lifecycle. During the design time, the enforcement process, will interpret the security policies generated by the risk assessment and enforce them as applicable rules to be checked against the TPM attested values. Likewise, during the run-time risk assessment, risk goals alongside with actual run-time attested values are fed into the risk quantification engine in reverse chaining inference mode, in order to assess the actual threat levels of the deployed system and mitigate any deviations from the expected results.

The Drools Expert business rule management system (BRMS) will be used to assist in the development of the policy enforcement mechanism. More specifically, the Drools system, is a rule-based system, that stores rules in order to be able to interpret input data in a useful way and produce verdicts on how the data should be handled. The Drools system uses an improved implementation of the Rete algorithm for pattern matching between the input data and the patterns that exist in the rules.

3.4.1 Design Time & Run Time Enforcement

In our case, we are going to use the Drools system in order to create enforceable mitigation controls both during design- and run-time. For the former, the system will propose several **controls that will map to a subset of properties that should be attested by the end device in order to reach the desired level of assurance**. On the other hand, during the run-time, the mechanism will collect data and evidence from the end device in order to calculate the actual run time threats. Depending on the results, there could be enough indications that may support a dynamic update of the design-time pre-defined security policies that will enforce new properties to be attested by the TPM.

The risk assessment and enforcement process is briefly demonstrated in Figure 9 with the green coloured components being part of the enforcement mechanism. More specifically, during the design-time, the enforcement mechanism is responsible for interpreting the risk levels produced by the risk quantification engine and assessing if they are acceptable or not. If the risk level is above a certain threshold, then the model is deemed as not acceptable and the quantification engine is called again in backward chaining inference mode, in order to produce a new model that contains new controls that directly map to a sub-set of the already identified properties of the system. This procedure will be run iteratively until the risk levels are below the acceptable threshold. When this is achieved, then the minimum set of needed properties to be attested are selected and enforced. The ultimate target of this process is to retain a specific level of assurance (risk level) while, at the same time, safeguarding the privacy of the attested device.

Furthermore, during the run-time of the system, there will be an extra set of actions that will check the attested values in order to assess the actual risk level of the device. The risk enforcement process is responsible for checking these attested values, collect any evidence

related to any found divergences and analyse this evidence in order to provide useful information to the risk analysis process. This process will lead to another iteration of the design-time process that will create updated enforceable and attestable properties.

However, the minimization of different risks may end-up in conflicting enforcement of controls. The enforcement mechanism will provide indications of salience in order for the quantification engine to end up in an optimal configuration that has to be enforced. Furthermore, the new "optimal" configuration, will be validated once again through the aforementioned risk assessment process, in order to be tested for any unchecked threats and assert that the threat levels of the new model are consistent with the targets of the implementation.

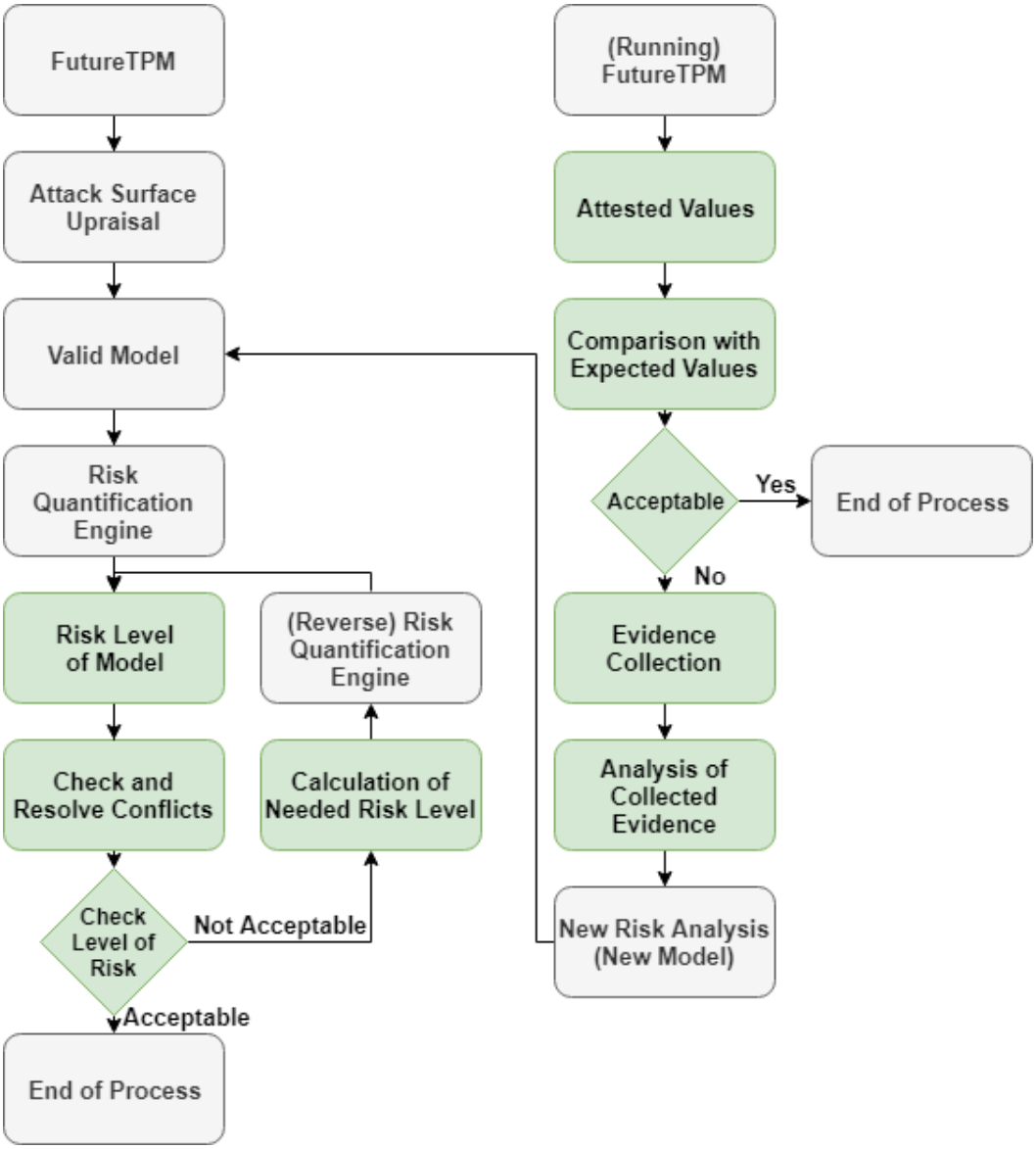

Figure 9: Design Time & Run Time Risk Analysis

3.4.2 Policy Based Access Control

A central point of the enforcement mechanism are the policies, which should be utilized and managed in a proper way. Furthermore, determining what policies, and classes of policies, need to be expressed and enforced within our architecture is critical and will be documented in the context of WP3 (D3.2). It is imperative that we are able to express policies that: (i) when enforced, mitigate the risks of the **safety** and **security** of the critical systems we wish to

compose, (ii) regard properties that can be attested to by the resource constrained embedded components of which systems are composed, (iii) safeguard the **privacy** of attesting devices by specifying the general principles for attestation data protection, and (iv) specify the type of evidence to be collected from a system, in case it fails to attest some of its properties, so as to perform a more in-depth investigation of the system's behavior to detect if any type of malware is resident to the program and data memory. The **Policy Models and Specification** acts as the interface between the outputs of the RA framework and the FutureTPM Policy Enforcement Toolkit.

To that end, FutureTPM will consider the usage of **Policy-Based Access Control (PBAC) models** that mandates access to system resources by reading a pre-specified set of rules and policies and provides a great level of granularity over the access rights of entities to resources. The architecture of this system is shown in Figure 10. The two basic entities of the proposed system are the **Policy Decision Point (PDP)** and the **Policy Enforcement Point (PEP)** like the ones found on RFC 2753 Framework for Policy-based Admission Control [85] on which the system is based. The PDP is a remote trusted server that will perform the risk analysis process and generate the properties to be attested, whereas the PEP entity resides within each TPM and it is materialized in the form of the TPM2.0 Enhanced Authorization (EA) in combination with the tracing from the eBPF system.

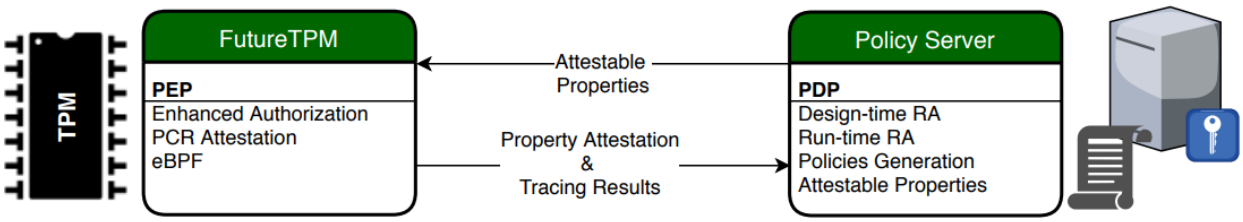

Figure 10: Policy Enforcement Architecture

In the PBAC model the PEP is responsible for managing all local requests and forwarding them to the PDP. The PDP will then make a decision to grant or deny permission based on some predefined policies that has stored and send the final verdict for the PEP to enforce. The FutureTPM will attempt a slightly different approach, as the end devices will have more authority since they will be able to grant or deny access to resources based on installed TPM2.0 Enhanced Authorization policies. The enhanced authorization function is based on policies that can check many attributes of the system in order to authorize inbound commands. The main metric that the TPM has in order to check the system are the PCR values that store a hash-based system status checksum. When PCR checks or other installed checks fail, then the TPM will have to resort to the PDP policy server to assess the situation. The PDP policy server will respond with new and updated policies to be enforced, that where generated with the run-time process we described before. The TPMs should have enough flexibility to manage most normal use cases and send requests to the PDP server only on unknown scenarios and/or unexpected check values.

Aside from the aforementioned special requests the policy server might receive, it should also receive periodical attestations from each device in order to assess its risk status. When an attestation arrives at the policy server, it starts a run-time risk assessment process and it generates a possibly updated set of policies that are translated to new attestable properties. If the threat level of the TPM is below the specified threshold, then the process finishes as there is no need for any changes, on the other hand, if the threat is not acceptable, new policies are dynamically generated in order to mitigate the threat. The properties that correspond to these policies are forwarded to the end device, where they are enforced through enhanced authorization functionality.

The server should provide an end point where a security expert can configure it. It is through this interface that the design-time risk analysis will be instrumented. The administrator has the ability to create new policies, and import models to the system, to generate updated enforceable properties that will be pushed to all the relevant end devices. This way the system will have a dynamic set of policies and will be resilient to future threats. The server could provide the following functionalities:

- **Attack Surface Appraisal Tool:** The tool that will support the security analyst in developing the first valid model of the system.
- **Model Validation:** A tool to automatically check the models before they are fed into the quantification engine. The main purpose of this tool is to make sure that the model is compliant with what the quantification engine expects and also find possible logical errors in the model.
- **Risk Quantification Engine (forward and backward chaining modes):** This engine utilizes the Drools Expert System in order to assess the threat level of models (forward chaining mode) and to generate new models based on threat level goals (backward chaining mode). Each model contains policies to be enforced on the end devices.
- **Manual Conflict Resolution:** A tool to help security analysts in resolving conflicts found within models.
- **Automatic Assessment of Attestations and Evidence sent from the TPM:** The TPM will send attestations of its state periodically and also send any related collected evidence when appropriate. This tool will try to translate this data to help the security analyst understand it.
- **Tools to support the analysis of divergences from the expected attested properties:** This tool will propose changes to mitigate any deviations found. The aim of this tool is to support the security analyst in creating a new model that contains updated policies which take into account the newfound attestation divergences and minimize the related threats.

Chapter 4 Risk Assessment of the Reference Scenarios

Within the context of FutureTPM project, the Risk Assessment framework will be developed and tailored to both the security requirements of the whole TPM-based system (TPM and the host device) but also of the targeted ICT deployments through the three envisioned use cases providing a risk quantification methodology which is model-driven. As a starting point we will focus on three main TPM functionalities, one per scenario. More specifically, the three reference scenarios will focus on the **sealing, the Direct Anonymous Attestation (DAA) and the key creation and storage**, respectively. These identified functionalities and the TPM commands supporting these functionalities are the baseline to help us **a)** create a concrete threat model (in the context of WP3) and **b)** trace these identified commands for possible vulnerabilities. In the next deliverables of WP4, we will expand this risk assessment methodology based on the results of QR cryptographic algorithms identified and developed in the context of WP2. A list of the possible TPM commands per reference scenario, and the broken-down TPM commands (as identified in Table 16 of D1.2 (red colour)), that are of interest for the risk assessment are presented in **Table 3** below.

Table 3: TPM commands used for the risk assessment

Command	Reference Scenario 1 – "*Secure Mobile Wallet and Payments*"	Reference Scenario 2 – "*Personal Activity and Health Kit Data Tracking*"	Reference Scenario 3 – "*Device Management*"
TPM2_ActivateCredential		√	√
TPM2_Certify		√	√
TPM2_CertifyCreation			
TPM2_Commit		√	
TPM2_Create	√	√	√
TPM2_CreatePrimary	√		√
TPM2_EC_Ephemeral			
TPM2_ECC_Parameters			
TPM2_ECDH_KeyGen			
TPM2_ECDH_ZGen			
TPM2_EncryptDecrypt	√		
TPM2_EvictControl	√		√
TPM2_FlushContext	√		√

Command	Reference Scenario 1 – "*Secure Mobile Wallet and Payments*"	Reference Scenario 2 – "*Personal Activity and Health Kit Data Tracking*"	Reference Scenario 3 – "*Device Management*"
TPM2_GetRandom	√		
TPM2_Hash		√	
TPM2_Load	√	√	√
TPM2_MakeCredential			√
TPM2_PCR_Extend	√		√
TPM2_PolicyGetDigest	√		√
TPM2_PolicyPCR	√		√
TPM2_RSADecrypt			√
TPM2_RSA_Encrypt			
TPM2_Sign		√	
TPM2_StartAuthSession	√		√
TPM2_Unseal	√		√
TPM2_VerifySignature		√	
TPM2_ZGen_2Phase			

The following subsections thoroughly describe each reference scenario by identifying how the risk assessment methodology will be applied based on the exact needs of each use case, the type of TPM environment that will be tested in each scenario and all the identified TPM commands that will used for the implementations.

4.1 Reference Scenario 1 – *Secure Mobile Wallet and Payments*

As described in Chapter 3, the first step in a holistic risk assessment process is to identify all the assets. Towards this direction and to have an overall view of this scenario, we list below all the assets, as have already identified in D1.1:

- **Asset 1:** Web Server - A REST API web server developed in Django Web Application Framework (Django REST).
- **Asset 2:** Mobile Device - An Android device that contains the mobile wallet FreePOS native mobile application.
- **Asset 3:** TPM - A TPM chip hosted in the mobile device.

In the context of FutureTPM, we especially focus on the risk assessment of the TPM and TSS. To this end, we first need to identify which TPM commands are going to be used, in order to create a threat model and trace them and identify possible vulnerabilities. In this use case, the

hardware-based TPM environment will be employed, as has already been discussed in D1.1 and we will focus on the **sealing feature**. In the future, we will probably update the TPM commands (if needed) based on the output of D2.2. The Asset 2 is an Android device with the mobile wallet FreePOS application that also hosts a TPM chip. The FreePOS application authentication is based on the OAuth 2.0 scheme. OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner. The third party then uses the access token to access the protected resources hosted by the resource server. The current implementation of the FreePOS application stores two different tokens necessary for the authentication on the device's main storage:

- the FreePOS token that authenticates between the client and the business logic; and
- the bearer token required to authenticate with the Payment Card Industry (PCI) compliant services.

However, if an attacker acquires access to the device file system or memory space, the attacker could be able to steal these credentials, impersonate the user and compromise the overall payment service.

Sealing (or local attestation) is the process when data (e.g. data blobs or cryptographic keys) is locked to PCRs. In the frame of secure mobile wallet and payments scenario both the FreePOS and the bearer tokens need to be secured. A common practice is to protect sensitive data with a randomly generated password, then seal that password data blob to PCR values. This scheme has the advantage of having an easy password less access to the key when the host is in a good know state. Additionally, by storing the password offline, the user has a recovery solution, access to the key is always granted when the password is entered manually. The same approach will be followed in this reference scenario. It is common to seal a data value to PCR values so that the data value can only be recovered if the platform has booted in a satisfactory way. Sealing data means that data is encrypted. Unlike binding data which is bound to a specific key, the data is bound to the TPM and the systems configuration when sealing data. In other words, data sealed by one TPM also has to be unsealed by the same TPM [84]. For performance reasons and due to the limitation of the size of sealed data we will follow the password approach and we will seal a password that will create a symmetric key to protect the two tokens. The work in [86] shows the performance overhead of sealing and unsealing data, as the input data size increases.

4.1.1 TPM Commands

A session is an internal TPM object that encodes an authorization value. First, we need to extend PCR with a value and create a trial session object with the TPM2_StartAuthSession command. Trial policy sessions are neutered policy sessions. They can't authorize any actions, but they can be used to generate policy digests before creating entities. Then the user executes the command TPM2_PolicyPCR, passing in again the handle of the trial session and the PCRs selected, and the policy_digest is calculated. This command updates a folding hash field in the session object called policy_digest. The policy_digest value can be read through the TPM2_PolicyGetDigest. Finally, we plug the policy digest into authPolicy field and pass it to the TPM2_Create to seal the data blob. After the creation, the object needs to be explicitly loaded with the TPM2_Load. A possible alternative is to use the TPM2_CreateLoaded command to seal and load the object, however this is something that needs further investigation. The aforementioned procedure is depicted in Figure 11.

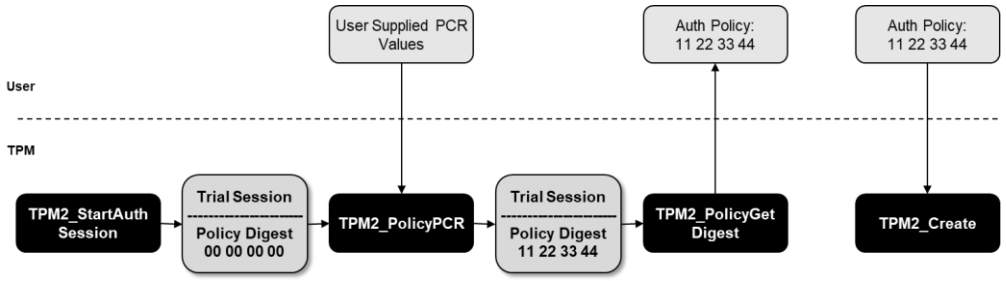

Figure 11: Calculating the authPolicy digest using a trial session [87]

Once an object is protected with an authPolicy, it can only be used with a session that encodes the expected digest. We create a new session object using TPM2_StartAuthSession and not a trial session. A call to TPM2_PolicyPCR hashes the contents of the actual PCR values into the session policy_digest field. Finally, when we call TPM2_Unseal, the TPM evaluates the session's policy digest and compares it against the expected authPolicy digest. If they match, the sealed data is returned to the user, otherwise, it fails with an error. This procedure is depicted in Figure 12.

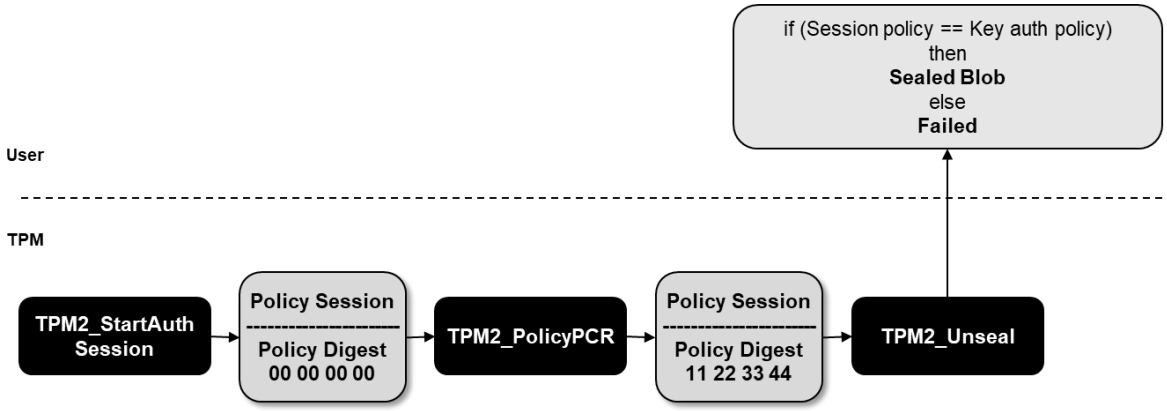

Figure 12: Policy session or Enhanced Authorization (EA) [87]

Table 4 and **Table 5** below summarize the core TPM commands that will be investigated in the context of this reference scenario.

Table 4: TPM commands for sealing a password and encrypting the tokens, in sequence

Command	Description
TPM2_PCR_Extend	Extend PCR with a value.
TPM2_GetRandom	Randomly generated password.
TPM2_CreatePrimary	Primary key necessary to create the objects.
TPM2_StartAuthSession	Create a trial session in order to calculate a policy.
TPM2_PolicyPCR	The auth policy digest of expected PCRs calculated using a trial session. User defined PCR values that must be present at the time of the unseal operation.
TPM2_PolicyGetDigest	Reads the policy digest value.

Command	Description
TPM2_Create	Protects the password with an auth policy. Seals that data blob (the password) to PCR values. Both the sensitive data (the password) and the policy are passed to the TPM2_Create. The PCR values represent the environment in which the object was created.
TPM2_Load	Load the created object in TPM, since it is not loaded automatically.
TPM2_EvictControl	Make the Transient Object into a Persistent Object.
TPM2_Create	Creates a password-protected symmetric encryption key.
TPM2_Load	Load the created object in TPM, since it is not loaded automatically.
TPM2_EvictControl	Make the created symmetric encryption key into a Persistent Object.
TPM2_EncryptDecrypt	Encrypt the two tokens with the password-protected symmetric key.
TPM2_FlushContext	Unload the primary.

Table 5: TPM commands for unsealing the password and decrypting the tokens, in sequence

Command	Description
TPM2_StartAuthSession	Start a new authorization (policy) session.
TPM2_PolicyPCR	Assertion that a selected set of PCRs have a specific value. Hashes the contents of the actual PCR values into the session policy digest field.
TPM2_Unseal	TPM evaluates the session's policy digest and compares it against the key's expected authPolicy digest. If they match, the sealed data is returned to the user (the password), otherwise, it fails with an error.
TPM2_FlushContext	Unload the session.
TPM2_EncryptDecrypt	Decrypt the encrypted tokens with the password-protected symmetric key.
TPM2_EvictControl	Flush the persistent keys.

4.2 Reference Scenario 2 – *Personal Activity and Health Kit Data Tracking*

The assets belonging to the Personal Activity and Healthkit Data Tracking (S5 Tracker) scenario, as identified in D1.1 are the following:

- **Asset 1:** Personal Web Application (S5PersonalTracker or S5DataAnalysis) – A personal web application, residing in the computer of a user (an individual) which is used to push data (in the case of individuals), to the S5Tracker Analytics Engine. In view of the TPM commands section below, this entity is referred to as the "host", who wishes to send data in an anonymous manner.
- **Asset 2:** TPM - A software TPM bundle hosted in the device hosting Personal Web Application, which is responsible for signing the messages of the host. The TPM alongside with the "host" are referenced thereafter as the "platform".
- **Asset 3:** Web Server and Analytics Engine (S5Tracker Analytics Engine) - A REST API web server developed in Django Web Application Framework (Django REST). In view of the TPM commands section below, this entity is referred to as the "verifier", who tries to verify that the "host" is a legitimate machine that can perform the data upload operation.

As already introduced in deliverable D1.1, this reference scenario demonstrates the case of the software-based TPM and attempts to implement data transfer preserving **anonymity** and **privacy**, using the **Direct Anonymous Attestation (DAA)** method. In this case, the TPM commands which are presented in the following sub-section relate to the DAA scheme.

Currently, the overall S5Tracker infrastructure relies on the OAuth 2.0 scheme for authentication between the different clients that provide or retrieve data from the system. This scheme, however, as used in the infrastructure, does not allow for privacy preservation and anonymity as necessary for the business evolution of the S5 Tracker. Data are indeed anonymised and obfuscated prior of being uploaded to the platform. Nonetheless, this method has two main drawbacks; firstly, privacy and anonymity is not preserved in case someone gains access to the S5Tracker Analytics Engine, and secondly, data may lose part of their value as the current obfuscation methods may lead to inaccurate assumptions (or to better put it, not highly accurate results), as the higher degree of anonymity, the more obfuscation filters are applied.

The first issue can be tackled through the use of DAA in order to offer a higher level of trust and privacy to the different individuals who choose to provide their data through the S5Tracker implementation. More specifically, employing a DAA mechanism, will allow the verifier machine (e.g. the S5Tracker Analytics Engine), to accept connections (and, thus, data push/pull requests) from remote machines (e.g. the hosts), which either belong to individuals or to analysts, without being able to distinguish and identify to whom they belong to. In this case, stronger cryptographic mechanisms are included in the overall picture, while the authenticity of the host machine is also examined by the TPM, which is something that is not supported by the current OAuth 2.0 scheme.

It needs to be noted that the envisaged scenario also expands to one step further, by having TPMs of other hosts (of the ones that will retrieve data – e.g. belonging to data analysts) to also examine if the verifier (S5Tracker Analytics Engine) is a trusted machine (booted in the correct order and being in a good state). However, this implies the insertion of new assets and methods into the scenario which shall follow the implementation of the DAA execution (in terms of the demonstrator – but are in reverse order in the logical flow, meaning that the S5PersonalTracker should first trust the S5Tracker Analytics Engine (through a proxy third party source), and then establish an anonymous connection). Nevertheless, this is not in the core focus of the scenario describe in this section.

4.2.1 TPM Commands

As discussed in the exemplary paper of [88], a DAA signature can be used to sign an arbitrary message given by the host such as a nonce from a verifier and a public key created by the host. In this case, DAA can be used to construct anonymous authentication by combining it with TLS [89], which prevents the sharing of credentials under the assumption that users cannot extract secret keys from TPMs.

The above-mentioned example is highly relevant to the current use case, as for the S5Tracker implementation the desired security properties for DAA are **anonymity, unforgeability and non-frameability** [88] of the messages (e.g. the data) to be send to the verifier. As such, in this DAA scheme, there are four types of parties: TPM **Mi** and host **Hj** constituting a platform, issuer **I** and verifier **V**, which are illustrated in the following picture.

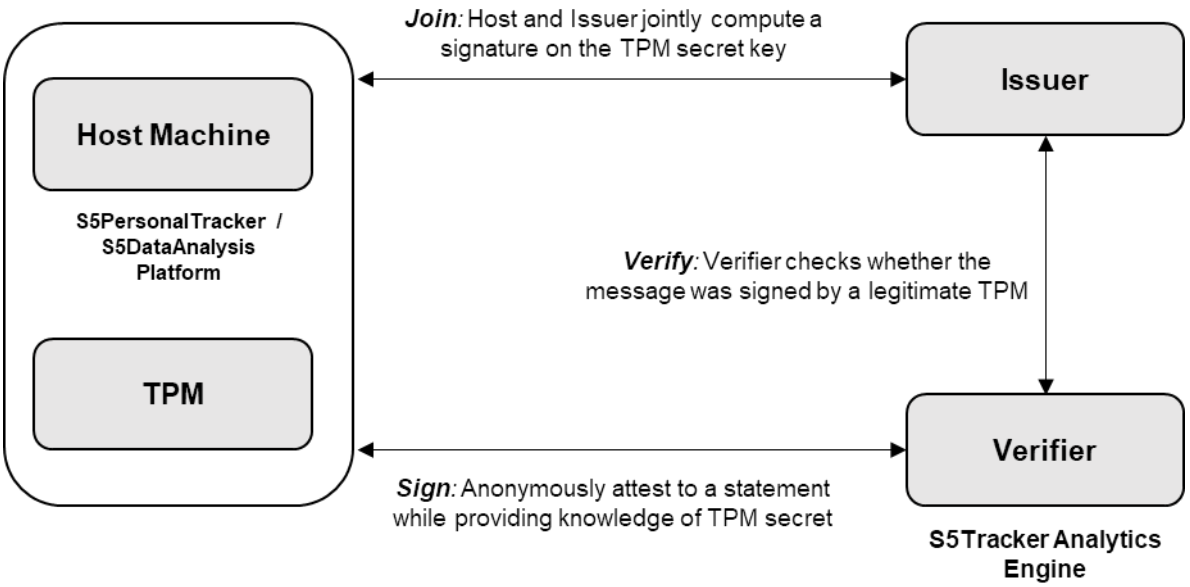

Figure 13: DAA Protocols [90] and the case of the S5TRacker

The DAA scheme consists of three algorithms **Setup, Verify and Link, and two protocols Join and Sign**. How to implement these algorithms and protocols by using the TPM2.0 commands are explained thoroughly in [88]. These commands are listed in the following tables. In our case, all client platforms (S5PersonalTracker) want to provide a cryptographic assertion that are genuine devices with certified characteristics. By using DAA in this use case we can avoid profiling user's online activities through analysis of their signatures and thus trace back these host devices to real users.

As shown in the figure above, the host platform must first receive credentials from an Issuer (who has already implemented the Setup algorithm (not shown in the figure) to generate the public and private keys). This is done by the Join protocol. This operation needs to be performed a single time, as then the platform is listed as valid and is allowed to become a member.

Table 6: TPM commands used in the DAA-Join Protocol

Command	Description
TPM2_Create	Creates a restricted key blob.
TPM2_Load	Load the created object in TPM, since it is not loaded automatically.
TPM2_Hash	Compute a hash digest with different message lengths. The results of the hash will be used in a signing operation that uses a restricted signing key and the ticket returned by this command can indicate that the hash is safe to sign.
TPM2_Commit	TPM Join by commit an ephemeral secret for signing.
TPM2_Sign	Host Join by generating a signature.
TPM2_ActivateCredential	Enables the association of a credential with an object in a way that ensures that the TPM has validated the parameters of the credentialed object. Allow the DAA issuer to authenticate the public key of a TPM.

After the Join protocol, once the platform (S5 Personal Tracker) needs to send data to the S5 Analytics, Engine, it will use the Sign protocol to sign these messages in order to be able to prove to the S5 Analytics Engine that it is a genuine platform, without however disclosing its identity.

Table 7: TPM commands used in the DAA-Sign Protocol

Command	Description
TPM2_Commit	Performs the first part of an ECC anonymous signing operation.
TPM2_Create	Generate the new authentication key pair.
TPM2_Load	Load the created object in TPM, since it is not loaded automatically.
TPM2_Certify	Proves that an object with a specific Name is loaded in the TPM.
TPM2_Sign	ECC anonymous signature. The same secret should be used in the TPM2_Commit command.

Finally, once messages are retrieved by the S5 Tracker Analytics Engine, the verifier checks the validity of those messages and decides if they are acceptable or not. Based on the DAA, any verification software on the host device can be used for this purpose. However, in the context of risk assessment and completeness proposes we will examine to use the TPM on host device for verification. **Table 8** summarizes the TPM commands for verification.

Table 8: TPM commands used in the Verify Protocol

Command	Description
TPM2_Commit	Performs the first part of an ECC verify operation.
TPM2_Create	Generate the new authentication key pair.
TPM2_Load	Load the created object in TPM, since it is not loaded automatically.
TPM2_Certify	Proves that an object with a specific Name is loaded in the TPM.
TPM2_VerifySignature	Validates the message with the message digest passed to the TPM.

4.3 Reference Scenario 3 – *Device Management*

In the Device Management use case, there are three main assets:

- **Asset 1:** network devices – routers, switches etc., which are processing network traffic of users;
- **Asset 2:** user devices and servers – endpoints of a communication established with the purpose of accessing data from a repository;
- **Asset 3:** NMS (Network Management System) – centralized system which is responsible for managing network devices in the infrastructure.
- **Asset 4:** TPM - Virtual TPM.

It leverages the virtual-based TPM environment to protect the communication keys that are used between network devices and the NMS, and to prevent that user data is processed by compromised devices. Currently, identification of network devices is weak, due to the fact that the key used to establish a secure channel is stored unprotected in the disk, and can be easily moved to another device (possibly controlled by an attacker).

With FutureTPM, identification will be done with a TPM key which never leaves the TPM in plain text. **The focus of the risk assessment will be to ensure that the key will be generated with safe parameters (key length, algorithm) and that it will be used only for the purpose of performing management tasks requested by NMS**. A second aspect of the risk assessment is the **protection of user data** while in transit over the network infrastructure. At any point in time, it could be possible that one or multiple network devices become corrupted and may try to steal or modify user data. It will be the responsibility of the NMS to determine the risk associated with using a specific device, depending on the device integrity status, and to decide whether or not the traffic should be diverted to a more trustworthy device. The last aspect of risk assessment to consider is the **protection of information stored in the NMS** that will affect the behaviour of the whole infrastructure. The NMS will be responsible to manage device credentials (e.g. TPM Endorsement Credential), and sign certificates that each device will provide to the NMS during the establishment of a trusted channel; the NMS will also be responsible of managing whitelists in order to determine whether the TPM key generated by the devices is associated to a good software configuration; lastly, the NMS will also enforce the policies defined by the network administrator by instructing the network devices to divert the traffic if some of them become compromised. Although it could be possible to protect this

sensitive information by adding a TPM to the NMS, the focus of the project will be to protect the TPM keys in network devices.

4.3.1 TPM Commands

One fundamental property of the TPM is that it is tamper resistant. Generating a key inside a TPM ensures that nobody, not even an attacker with physical access to the device, will be able to extract that key. A key can only be used by invoking a method exposed by the TPM. In the context of this use case, a TPM key will be used to establish a trusted channel with the NMS, so that the NMS can have the proof that the communication originated from a specific device (the proof can be added for example in a X.509 certificate extension defined by the TCG which is called *Subject Key Attestation Evidence* or *SKAE*). The second fundamental property of the TPM is that it can securely store the current system state in its Platform Configuration Registers (PCRs) and it will allow certain crypto operations to be performed with TPM keys only if the current state is the same as when the TPM key was created. For this use case, network device will implicitly prove to the NMS that a device is in the expected state simply by performing the trusted channel handshake with the NMS. If the state is different, the device TPM will not perform the crypto operations and the NMS can conclude that the device is in a different state (it could have been compromised).

The device management demonstrator will use TPM commands similar to those listed in **Table 4** and **Table 5**, **Table 9** and for the Secure Mobile Wallet demonstrator, with a few additions. The complete list of TPM commands used by the device management demonstrator is reported below.

Table 9: TPM commands used during setup phase, in sequence

Command	Description
TPM2_CreatePrimary	Create the primary key.
TPM2_Create	Create an Attestation Key (AK).
TPM2_MakeCredential	Create an activation credential (symmetric key to encrypt AK certificate).
TPM2_ActivateCredential	Retrieve an activation credential (symmetric key to decrypt AK certificate).
TPM2_StartAuthSession	Create a trial session in order to calculate a policy.
TPM2_PolicyPCR	Update the policy digest with good PCR values.
TPM2_PolicyGetDigest	Reads the policy digest value.
TPM2_Create	Generate a new TLS key to establish a trusted channel and associated it to the policy returned by TPM2_PolicyGetDigest.
TPM2_Load	Load the AK and TLS keys.
TPM2_Certify	Certify the TLS key with the created AK.
TPM2_Create	Generate random data to be used as HMAC key and associated it to the policy returned by

Command	Description
	TPM2_PolicyGetDigest.
TPM2_EvictControl	Make the primary key persistent to improve performance.
TPM2_FlushContext	Unload the primary, AK and TLS keys.

Table 10: TPM commands used during runtime, in sequence

Command	Description
TPM2_PCR_Extend	Extend PCR with digests of loaded software.
TPM2_StartAuthSession	Start a new authorization (policy) session.
TPM2_PolicyPCR	Update the policy digest with current PCR values.
TPM2_Load	Load the HMAC key, since it is not loaded automatically.
TPM2_Unseal	TPM evaluates the session's policy digest and compares it against the key's expected authPolicy digest. If they match, the sealed data is returned to the user, otherwise, it fails with an error.
TPM2_Load	Load the TLS key, since it is not loaded automatically.
TPM2_RSADecrypt	Perform a crypto operation during the establishment of the trusted channel.
TPM2_FlushContext	Unload the TLS key.

Chapter 5 Summary and Conclusions

This final section will act as a synopsis of this deliverable and will summarize its findings. The scope of this deliverable was to provide a detailed description of the Risk Management phase and more particularly the threat modelling and the risk assessment methodologies that will be developed within the project. FutureTPM Risk Assessment methodology aims to provide implementation guidance and deals with the identification of threats, determines their probability of occurrence, and their resulting impact. The assets of interest are the devices (e.g., processors, mobile devices, ASICS, etc.), hosting the TPM, the TPM itself and the TSS that provides all the underlying communication interfaces with the trusted hardware.

To reflect FutureTPM's Risk Assessment framework work and data flow and how provided assurance services are engrained at all stages of the development life cycle, this deliverable documents the envisaged architecture that consists of four (highly interdependent) phases, namely the **Design-time Risk Assessment**, the **Policy Modelling and Specification**, the **Run-time Risk Assessment** and the **Dynamic Update of the Policy Models**. This will serve as the guide for the development of the FutureTPM RA framework to be presented in the subsequent deliverables of WP4.

The **Risk Assessment** phase will be executed during both the **design-** and **run-time**, to address initially unknown threats. During the design-time, the risk assessment will be applied by security experts in order to provide a cartography of their TPM supported/enabled applications and services ecosystem. During the run-time, it will be applied towards the serialization of all information that is required to perform the re-calculation of relevant risks that may lead to a (possible) dynamic update of predefined security policies.

As the name suggests, the **Policy Modelling and Specification** phase contains all tasks related to the specifications of the FutureTPM composition control architecture which implements a policy-based approach for allowing proofs of a system's integrity based on the identified CPS configuration and execution properties to be attested. Defined policies must be *expressive*, *deployable*, and *enforceable* and may be dynamically updated if the attack graph (produced and maintained within the Risk Management phase) is amended with new types of vulnerabilities. Within the context of this phase, security properties formal verification and proofs of FutureTPM architecture (at design-time) are also performed. However, FutureTPM's run-time assurance framework must also address the dynamic runtime properties of all deployed CPSs which cannot be modelled completely at design-time; taking into consideration their changing control objectives and system configurations as well as the changing communication and computation resources and allocation.

In this context, we also have the **Policy Enforcement** phase where the defined access control policies and policy sets are made available to the deployed CPSs which can, then, start enforcing them and make decisions based upon the results of the remote property-based attestation processes. Remote property-based attestation denotes the attestation of high-level CPS configuration and execution properties as identified in the design-time risk assessment phase; i.e., current firmware version, presence of certain hardware properties, integrity of measurements, integrity of TSS software and control flow integrity, etc. Attestation will focus on the design space between the two extremes of software-only and hardware-based attestation to provide features such as "secure boot, tamper protection, isolation of safety-critical systems, and behavioural monitoring" against sets of vulnerabilities and risks as modelled in the Risk Management phase.

Another distinct feature of this phase is the provision of functionalities regarding modelling **policy dynamics and dynamic change of policies**. Considering the complexity of TPM-based systems where the deployed software may be updated to a more sophisticated version with new capabilities, the threat model has to take into consideration, e.g., zero day attacks that were not evaluated in the RA phase at design-time. On top of that, a decision about a system's integrity might not be sufficient to understand a system's behaviour when the attestation output is negative. In this case, another stream of assurance functionality is needed which entails a more in-depth

investigation to detect any cheating attempts or if any type of malware is resident to the system's program and data memory. This is the goal of the **Run-time Risk Assessment** phase: based on collected evidence regarding the behaviour of a system, runtime verification mechanisms are applied for monitoring and verifying both the execution behaviour of a single system as well as the communication patterns of a set of attesting systems against a set of specific requirements as identified in the design phase.

Overall, full transparency of all cyber-physical systems and how they operate in this complex environment, in terms of operational assurance, verification and security policy enforcement, mandates the continuous execution of the above described phases and functionalities throughout the whole life cycle of a TPM-based deployed ecosystem. This means that the FutureTPM's RA framework internal technical components run in parallel exchanging the necessary information for triggering actions, within the previously described assurance and risk management phases, in order to achieve its overall goal of providing "security-by-design" and during operation. **Given the overall description of FutureTPM's RA framework conceptual architecture and the dependencies and interactions between its internal components, the detailed documentation and implementation of each one of these integral components will follow in the context of WP4.**

Last but not least, this deliverable highlights the possible TPM commands that are going to be investigated per reference scenario, in order to detail how the risk assessment methodology will be applied based on the exact needs of each scenario. As a starting point for the modelling and tracing, the identified commands will be used, with a possible expansion based on the output of the implementation of the QR crypto algorithms in the various TPM environments (WP5).

Chapter 6 List of Abbreviations

Abbreviation	Translation
APT	Advanced Persistent Threat
BCC	BPF Compiler Collection
BRMS	Business Rule Management System
CAPEC	Common Attack Pattern Enumeration and Classification
CFI	Control Flow Integrity
CIA	Confidentiality, Integrity and Availability
CM	Countermeasure
COBIT	Control Objectives for Information and Related Technologies
COTS	Commercial of the Shelf
CPE	Common Platform Enumeration
CPS	Cyber Physical Systems
CRRA	Cyber Risk Remediation Analysis
CTSA	Cyber Threat Susceptibility Assessment
CVE	Common Vulnerability and Exposure
CVSS	Common Vulnerability Scoring System
DAA	Direct Anonymous Attestation
EA	Enhanced Authorization
eBPF	Extended Berkeley Packet Filter
ESE	Enterprise Systems Engineering
FAIR	Factor Analysis of Information Risk
GCC	GNU Compiler Collection
ICT	Information and Communications Technology
IEC	International Electrotechnical Commission
ISMS	Information Security Management System
ISO	International Organization for Standardization

Abbreviation	Translation
JIT	Just-in-Time
LEF	Loss Event Frequency
MAE	Mission Assurance Engineering
OCTAVE	Operationally Critical Threat, Asset, and Vulnerability Evaluation
PBAC	Policy-Based Access Control
PCI	Payment Card Industry
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PID	Process ID
PLM	Probable Loss Magnitude
QR	Quantum-Resistant
RA	Risk Assessment
RM	Resource Manager
RMF	Risk Management Framework
SDLC	System Development Life Cycle
SEI	Software Engineering Institute
TAB	TPM Access Broker
TARA	Threat Assessment & Remediation Analysis
TEF	Threat Event Frequency
TPM	Trusted Platform Module
TTP	Tactics, Techniques, and Procedure
VFS	Virtual File System
WP	Work Package

Chapter 7 Bibliography

[1] T. F. Consortium, “FutureTPM Reference Architecture,” 2018.

[2] T. F. Consortium, “D1.1 - FutureTPM Use Cases and System Requirements,” 2018.

[3] International Organization for Standardization, “ISO/IEC 25010:2011 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models,” 2011.

[4] L. Kohnfelder and G. Praerit, “The threat to our products,” 1999. [Online].

[5] R. S. Ross, “Guide for Conducting Risk Assessments,” NIST, 2012.

[6] J. A. Christopher, J. D. Audrey, “OCTAVE Criteria, Version 2.0,” *Software Engineering Institute ,* December 2001.

[7] MITRE, “Threat Assessment & Remediation Analysis (TARA),” *MITRE TECHNICAL REPORT,* October 2011.

[8] J. Freund, J. Jones, Measuring and Managing Information Risk: A FAIR Approach, Elsevier Inc, 2015.

[9] ISACA, “Understanding the Core Concepts in COBIT 5,” *ISACA JOURNA,* vol. 5, 2013.

[10] NIST, “Risk Management Framework for Information Systems and Organizations,” *NIST Special Publication 800-37,* December 2018.

[11] International Organization for Standardization, “ISO/IEC 27005:2018 Information technology -- Security techniques -- Information security risk management,” *International Standard,* July 2018.

[12] A. Shostack, in *Threat Modeling: Designing for Security*, Wiley, 2014, p. 78.

[13] L. Osterman, “Threat Modeling Again, STRIDE per Element.,” 2007. [Online]. Available: https://blogs.msdn.microsoft.com/larryosterman/2007/09/10/threatmodeling-again-stride-per-element/.

[14] S. Krishnan, “A Hybrid Approach to Threat Modelling A Hybrid Approach to Threat Modelling,” 2017.

[15] V. Saini, Q. Duan, and V. Paruchuri, “Threat modeling using attack trees,” *Journal of Computing in Small Colleges,* vol. 23, no. 4, pp. 124-131, 2018.

[16] C. Yue, B. Boehm and L. Sheppard, “Value driven security threat modeling based on attack path analysis.,” in *40th Annual Hawaii International Conference on System Sciences (HICSS 2007)*, 2007.

[17] “IBM's TPM 2.0 TSS,” IBM, [Online]. Available: https://sourceforge.net/projects/ibmtpm20tss/.

[18] “Intel TSS,” Intel, [Online]. Available: https://github.com/tpm2-software/tpm2-tss.

[19] “Microsoft TSS,” Microsoft, [Online]. Available: https://github.com/Microsoft/TSS.MSR.

[20] “Trousers TSS,” [Online]. Available: http://trousers.sourceforge.net/.

[21] “Java TSS,” [Online]. Available: https://sourceforge.net/projects/trustedjava/.

[22] W. Mao, Y. Fei and C. Chunrun, “Daonity: grid security with behaviour conformity from trusted computing,” in *First ACM workshop on Scalable trusted computing*, 2006.

[23] C. Shen, H. Zhang, H. Wang, J. Wang, B. Zhao, F. Yan, F. Yu, L. Zhang, M. Xu, “Research on trusted computing and its development,” *Science China Information Sciences,* vol. 53, no. 3, pp. 405-433, 2010.

[24] J. Shao, D. Feng and Y. Qin, “Type-based analysis of protected storage in the TPM,” in *International Conference on Information and Communications Security*, Beijing, China, 2013.

[25] Q. Zhang, S. Zhao, Y. Qin and D. Feng, “Formal analysis of TPM2.0 key management APIs,” *Chinese Science Bulletin,* vol. 59, no. 32, p. 4210–4224, 2017.

[26] W. Wang, Y. Qin,and D. Feng, “Automated proof for authorization protocols of TPM 2.0 in computational model,” *International Conference on Information Security Practice and Experience,* vol. 8434 of LNCS, p. 144–158, 2014.

[27] L. Chen and M. Ryan, “Attack, solution and verification for shared authorisation data in TCG TPM,” *International Workshop on Formal Aspects in Security and Trust,* vol. 5983 of LNCS, p. 201–216, 2009.

[28] J. Shao, Y. Qin, D. Feng, and W. Wang, “Formal analysis of enhanced authorization in the TPM 2.0.,” in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS ’15)*, Singapore, 2015.

[29] E. F. Brickell, J. Camenisch and L. Chen, “Direct anonymous attestation,” in *11th ACM Conference on Computer and Communications Security, CCS*, Washington, DC, USA, 2004.

[30] M. Backes, M. Maffei and D. Unruh, “Zero-knowledge in the applied Pi-calculus and automated verification of the direct anonymous attestation protocol.,” in *IEEE Symposium on Security and Privacy (S&P)*, Oakland, California, USA, 2008.

[31] E. Brickell, L. Chen and J. Li, “Simplified security notions of direct anonymous attestation and a concrete scheme from pairings,” *International journal of information security,* vol. 8, no. 5, pp. 315-330, 2009.

[32] L. Xi and D. Feng, “Formal analysis of DAA-related APIs in TPM 2.0,” in *In International Conference on Network and System Security, Lecture Notes in Computer Science*, vol. 8792, Xi'an, Springer, 2014, pp. 421-434.

[33] J. Camenisch, M. Drijvers and A. Lehmann, “Anonymous attestation with subverted TPMs,” in *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference*,

Santa Barbara, CA, USA, 2017.

[34] J. Camenisch, L. Chen, M. Drijvers and A. Lehmann, “One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation,” in *IEEE Symposium on Security and Privacy, SP*, San Jose, CA, USA, 2017.

[35] F. E. Allen., “Control flow analysis,” in *Proceedings of a symposium on Compiler optimization*, NY, USA, 1970.

[36] S. Micha and A. Pnueli, “Two approaches to interprocedural data flow analysis,” pp. 189-234, 1978.

[37] J. Newsome and X. S. Dawn, “Dynamic Taint Analysis for Automatic Detection, Analysis, and SignatureGeneration of Exploits on Commodity Software.,” *NDSS,* vol. 5, 2005.

[38] B. Chess and G. McGraw, “Static analysis for security,” *IEEE Security & Privacy,* vol. 2, no. 6, pp. 76-79, 2004.

[39] Y. Yang, H. Zhang, M. Pan, J. Yang, F. He, and Z. Li, “A model-based fuzz framework to the security testing of tcg software stack implementations,” in *International Conference in Multimedia Information Networking and Security (MINES’09)*, 2009.

[40] G. Tóth, G. Koszegi and Z. Hornák, “Case study: automated security testing on the trusted computing platform,” in *Proceedings of the ACM SIGOPS European Workshop on System Security (EUROSEC)*, 2008.

[41] P. Godefroid, M. Y. Levin and D. Molnar, “SAGE: Whitebox Fuzzing for Security Testing,” *Queue ACM,* vol. 10, no. 1, pp. 20-27, 2012.

[42] C. Cadar and S. Koushik, “Symbolic execution for software testing: three decades later,” *Communications of the ACM,* vol. 56, no. 2, pp. 82-90, 2013.

[43] M. Aizatulin, A. D. Gordon and J. Jürjens, “Computational verification of C protocol implementations by symbolic execution,” in *ACM conference on Computer and communications security*, Raleigh, 2012.

[44] S. Musavi and M. Hashemi, “A Threat Modeling Framework for Evaluating Computing Platforms Against Architectural Attacks,” *CoRR,* 2018.

[45] H. Almohri, L. Cheng, D. Yao and H. Alemzadeh, “On Threat Modeling and Mitigation of Medical Cyber-Physical Systems,” in *IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, Philadelphia, PA, 2017.

[46] L. M. Vaquero, L. Rodero-Merino and D. Moran, “Locking the sky: a survey on IaaS cloud security,” *Computing,* no. 91, pp. 93-118, 2011.

[47] Royal Holloway University of London, “Data sharing using Trusted Platform Module,” 2010.

[48] J. Di and S. Smith, “A Hardware Threat Modeling Concept for Trustable Integrated Circuits,” *IEEE Region 5 Technical Conference,* pp. 354-357, 2007.

[49] J. Di, “Towards Trustable Embedded Systems: Hardware Threat Modeling for Integrated Circuits,” 2008.

[50] S. Scott and J. Di, “Detecting Malicious Logic Through Structural Checking,” in *IEEE Region 5 Technical Conference*, 2007.

[51] D. Wang, T. Zhou,Y. Wu and W. Zhao, “Risk assessment model for trusted platform control module based on Bayesian network,” *Journal of Computer Applications,* 2011.

[52] S. Akleylek, N. Bindel, J. Buchmann, J. Krämer and G.A. Marson, “An Efficient Lattice-Based Signature Scheme with Provably Secure Instantiation,” *Progress in Cryptology AFRICACRYPT 2016 Lecture Notes in Computer Science,* vol. 9646, 2016.

[53] M. J. Kannwischer, A. Genêt, D. Butin, J. Krämer and J. Buchmann, “Differential Power Analysis of XMSS and SPHINCS,” in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2018.

[54] A. Aysu, Y. Tobah, M. Tiwari, A. Gerstlauer and M. Orshansky, “Horizontal side-channel vulnerabilities of post-quantum key exchange protocols,” in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, Washington, DC, 2018.

[55] A. Facon, S. Guilley, M. Lec'Hvien, A. Schaub, Y. Souissi, “Detecting Cache-Timing Vulnerabilities in Post-Quantum Cryptography Algorithms,” in *IEEE 3rd International Verification and Security Workshop (IVSW)*, 2018.

[56] RISQ Project, “Cache-Timing Vulnerabilities of NIST PQC Competitors,” [Online]. Available: www.risq.fr/?page_id=473&lang=en.

[57] J. Wichelmann, A. Moghimi, T. Eisenbarth and B. Sunar, “MicroWalk: A Framework for Finding Side Channels in Binaries,” in *ACSAC 2018*, 2018.

[58] J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi, “Test Vector Leakage Assessment (TVLA) methodology in practice,” in *International. Cryptographic Module Conference*, 2013.

[59] W. Lei, L. Wang, W. Shan, K. Jiang and Q. Li, “A Frequency-Based Leakage Assessment Methodology for Side-Channel Evaluations,” in *13th International Conference on Computational Intelligence and Security (CIS)*, Hong Kong, 2017.

[60] W. Xiaofeng, S. Kun, Z. Yong and R. Jia, “A Side-channel Attack on HotSpot Heap Management,” in *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2018.

[61] A. Kurmus, S. Dechand, R. Kapitza, “Quantifiable Run-time Kernel Attack Surface Reduction,” in *Proceedings of the 10th International Conference on Detection of Intrusions and Malware Vulnerability Assessment (DIMVA’14)*, 2014.

[62] K. Sergey, “Dynamic Tracing with DTrace SystemTap,” [Online]. Available: https://myaut.github.io/dtrace-stap-book/dtrace-stap-book.pdf.

[63] IOVisor, “BCC - Tools for BPF-based Linux IO analysis, networking, monitoring, and more,” 2018. [Online]. Available: https://github.com/iovisor/bcc#tools.

[64] BCC authors, “BPF Compiler Collection (BCC),” [Online]. Available: https://www.iovisor.org/technology/bcc .

[65] Trusted Computing Group (TCG), “TCG TSS 2.0 TAB and Resource Manager Specification,” 2018.

[66] TPM2-abrmd authors, “TPM2 Access Broker & Resource Manager,” [Online]. Available: https://github.com/tpm2-software/tpm2-abrmd.

[67] UBITECH, “OLISTIC Risk Assessment Platform,” [Online]. Available: https://olistic.io/.

[68] The JBoss Drools team, “Drools Expert User Guide v5.4.0.Final,” 2012. [Online]. Available: https://docs.jboss.org/drools/release/5.4.0.Final/drools-expert-docs/pdf/drools-expert-docs.pdf.

[69] I. Gaag and V. Lida, Principles of Expert Systems, Addison-Wesley, 1991, pp. 131-139.

[70] ETSI TS 102 165-1, “Methods and protocols; Part 1: Method and proforma for Threat, Risk, Vulnerability Analysis,” in *Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN)*, 2014.

[71] M. P. T. W. a. D. S. S. L. Szekeres, “Eternal,” in *IEEE S&P*, 2013.

[72] E. B. H. S. a. S. S. R. Roemer, “Return-oriented programming: Systems, languages, and applications,” in *ACM TISSEC, 15(1):2:1–2:34, 2012*.

[73] H. Shacham, “The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86),” in *ACM CCS, 2007*.

[74] R. R. H. S. a. S. S. E. Buchanan, “When good instructions go bad: Generalizing return-oriented programming to RISC,” in *ACM CCS, 2008*.

[75] T. Kornau., “Return oriented programming for the ARM architecture,” in *Master’s thesis, Ruhr-University Bochum, 2009*.

[76] A. F. a. C. Castelluccia, “Code injection attacks on Harvard-architecture devices,” in *ACM CCS, 2008*.

[77] M. B. U. E. a. J. L. M. Abadi, “Control-flow integrity: Principles, implementations, and applications,” in *ACM TISSEC, 13(1), 2009*.

[78] F. B. Cohen, “Operating system protection through program evolution,” in *Computer & Security, 12(6), 1993*.

[79] L. S. M. P. G. C. R. S. V. Kuznetsov, “Code-pointer integrity,” in *USENIX OSDI, 2014*.

[80] J. X. E. C. S. P. G. a. R. K. I. S. Chen, “Non-control-data attacks are realistic threats,” in *USENIX Security, 2005*.

[81] N. A. L. D. J.-E. E. T. N. A. P. A.-R. S. G. T. Tigist Abera, “C-FLAT: Control-Flow Attestation for Embedded Systems Software,” in *ACM CCS 2016*.

[82] AlephOne, “Smashing the Stack for Fun and Profit,” in *Phrack Magazine, vol. 49, no. 14, 1996*.

[83] L. D. A. D. A.-R. S. H. S. a. M. W. S. Checkoway, “Return-oriented Programming Without Returns,” in *ACM Conference on Computer and Communications Security, 2010*.

[84] W. Arthur, D. Challener, A Practical Guide to TPM 2.0 Using the Trusted Platform Module in the New Age of Security, Apress, 2015.

[85] R. Yavatkar, D. Pendarakis, and R. Guerin, “RFC2753, A Framework for Policy-based Admission Control,” [Online]. Available: https://tools.ietf.org/html/rfc2753.

[86] N. Santos, R. Rodrigues, K.P. Gummadi and S. Saroiu, “Policy-Sealed Data: A New Abstraction for Building Trusted Cloud Services,” 2012.

[87] TPM-JS, “Sealing Data and Keys,” [Online]. Available: https://google.github.io/tpm-js/#pg_sealing.

[88] K. Yang, L. Chen, Z. Zhang, C. Newton, B. Yang and L. X, “Direct Anonymous Attestation with Optimal TPM Signing Efficiency,” *IACR Cryptology ePrint Archive 2018,* vol. 1128, no. 2018.

[89] E. Cesena, H. Lohr, G. Ramunno, A.-R. Sadeghi and D. Vernizzi, “Anonymous Authentication with TLS and DAA,” *TRUST’10: 3rd International Conference on Trust and Trustworthy Computing,* vol. 6101 of LNCS, pp. 47-62, 2010.

[90] N. R. Luther, “Implementing Direct Anonymous Attestation on TPM 2.0,” Blacksburg, Virginia, 2017.

[91] Intel, “The TPM2 Software Stack: Introducing a Major Open Source Release,” August 2018. [Online]. Available: https://software.intel.com/en-us/blogs/2018/08/29/tpm2-software-stack-open-source.

Appendix A

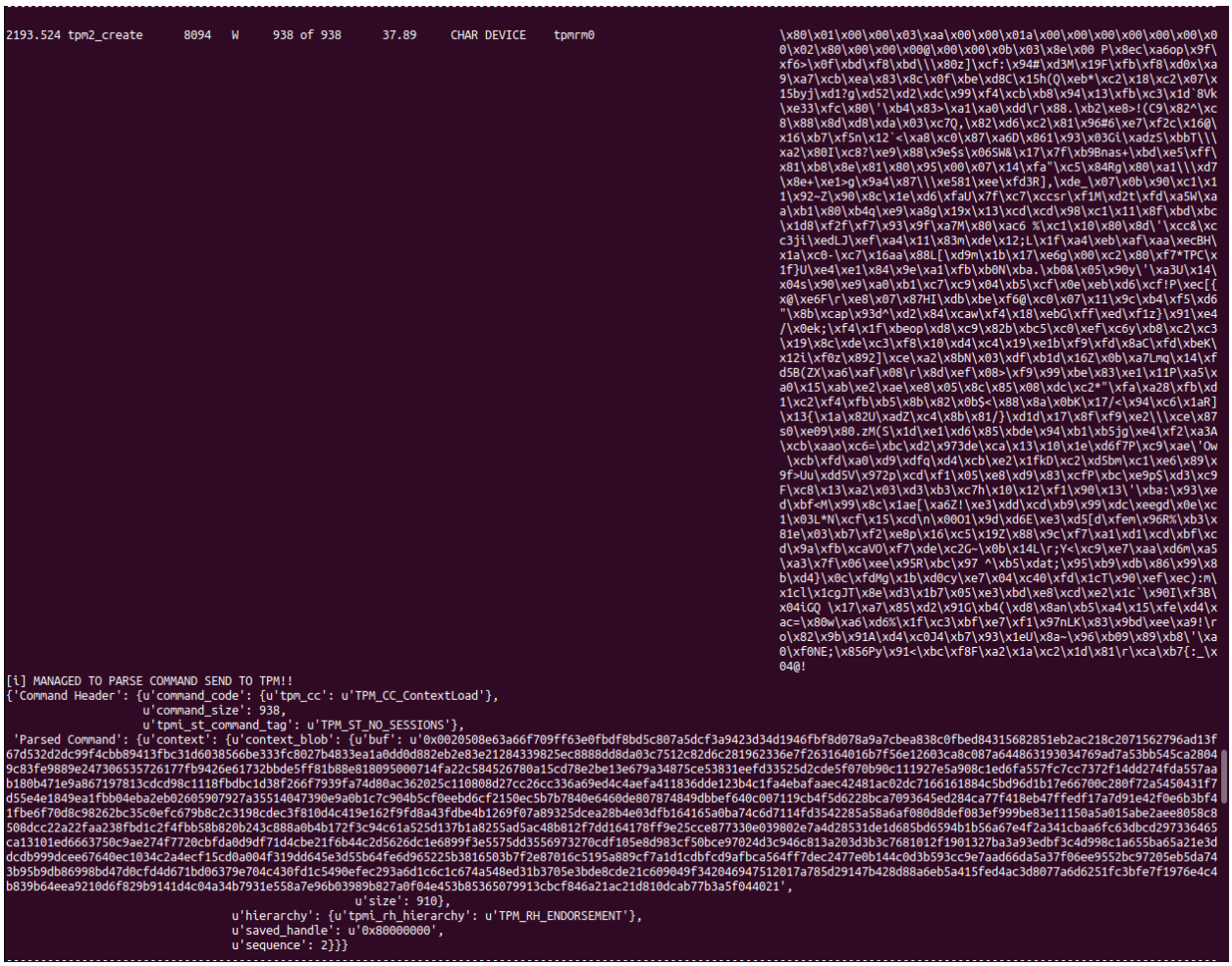

Figure 14: Trace Write TPM2_create TPM command | TPM_CC_ContextLoad

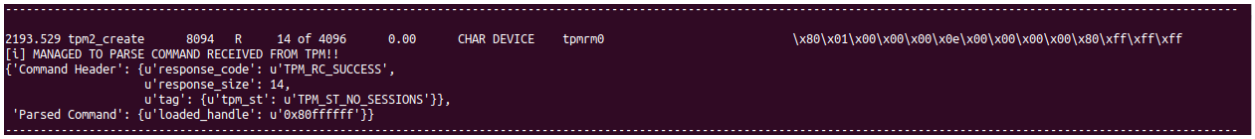

Figure 15: Trace Read TPM2_create TPM command | TPM_RC_SUCCESS

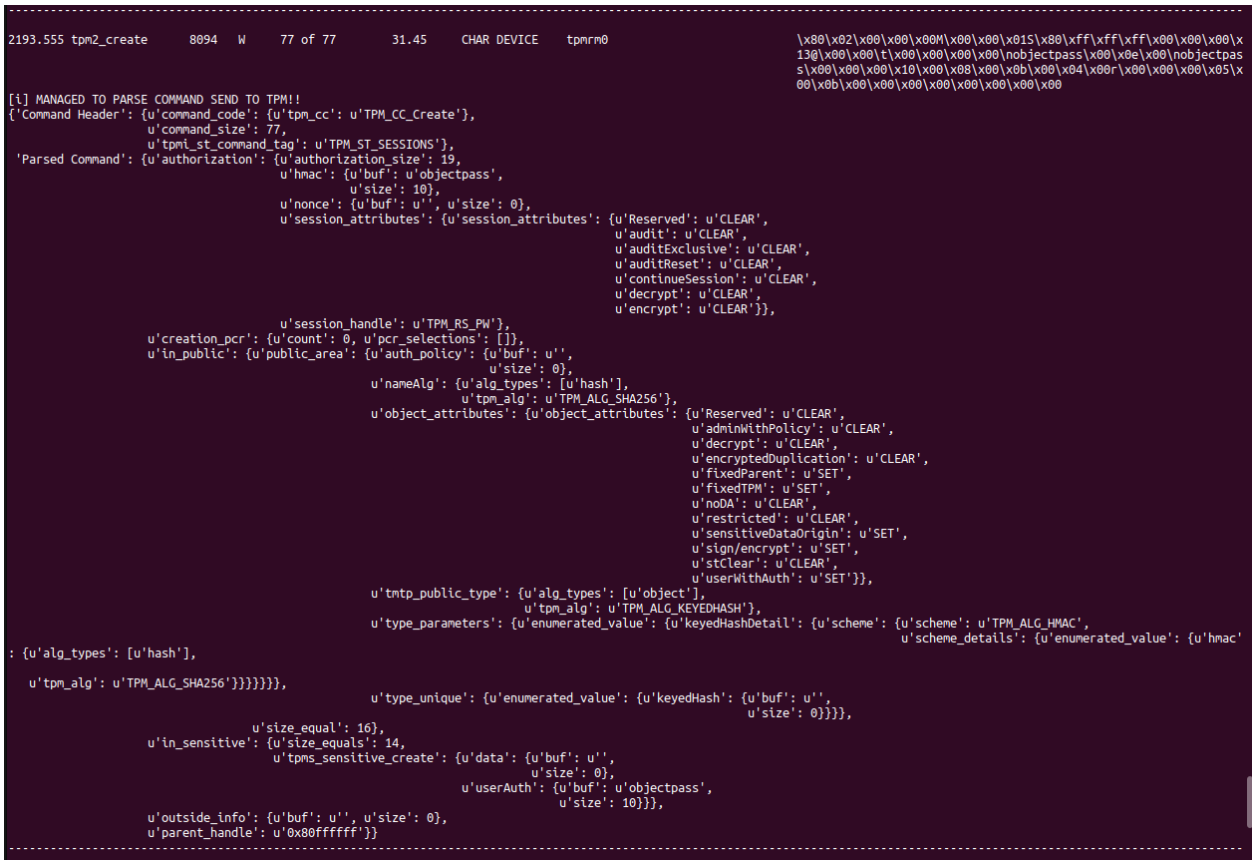

Figure 16: Trace Write TPM2_create TPM command | TPM_CC_Create

Figure 17: Trace Read TPM2_create TPM command | TPM_RC_SUCCESS